MASTER EN SCIENCES MATHÉMATIQUES

Etude du couplage entre un algorithme génétique et des méthodes d'optimisation locale

CRELOT, Anne-Sophie

Award date:
2011

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Faculté des Sciences

Etude du couplage entre un algorithme génétique et des méthodes d'optimisation locale

Promoteur :

Annick Sartenaer

Directeur :

Caroline Sainvitu

Mémoire présenté pour l'obtention

du grade académique de master en Sciences Mathématiques

Anne-Sophie CRÉLOT

JUIN 2011

Remerciements

Au début de ce mémoire, je souhaite remercier les personnes qui m'ont aidée durant toute la réalisation de ce travail. Leur soutien et leurs conseils m'ont été très précieux.

Je remercie d'abord Madame Annick Sartenaer d'avoir accepté d'être ma promotrice. Ses conseils et ses encouragements m'ont permis d'améliorer continuellement mon travail, mais m'ont surtout donné la confiance en mes capacités pour réaliser au mieux ce travail.

Je remercie ensuite Madame Caroline Sainvitu pour l'encadrement qu'elle m'a apporté, en particulier lorsque je travaillais à Cenaero. Ses explications et son aide se sont révélées très précieuses aussi bien au niveau du travail réalisé à Cenaero que dans le cadre de la rédaction.

Je remercie l'entreprise Cenaero de m'avoir permis de réaliser ce travail en proposant ce sujet mais surtout en m'accueillant durant plusieurs mois. Je remercie également les membres de l'entreprise pour leur accueil chaleureux.

Je remercie Monsieur Philippe Toint et Anke Tröltzsch de nous avoir permis d'utiliser l'algorithme BCDFO.

Je remercie également les membres du département de mathématiques pour leur disponibilité et leurs conseils.

Finalement, je tiens à remercier mes parents, ma soeur et mes amis pour le soutien et les encouragments qu'ils m'ont apportés, ainsi que pour leurs relectures attentives de ce travail.

Résumé

L'optimisation globale de fonctions coûteuses est le domaine de recherche dans lequel est réalisé ce travail. La résolution de tels problèmes intéresse fortement le monde de l'industrie qui manipule de plus en plus des fonctions de ce type suite à l'utilisation de simulations haute fidélité réalisées sur ordinateur. Ce mémoire a été réalisé en partenariat avec l'entreprise Cenaero.

L'objectif de ce mémoire est d'appliquer des algorithmes hybrides pour optimiser les fonctions coûteuses. Un algorithme hybride résulte de la combinaison entre une méthode de recherche globale, typiquement un algorithme génétique, et une méthode de recherche locale. Il existe de nombreuses façons de construire un algorithme hybride, à cause du choix de la méthode locale utilisée mais aussi en fonction de la manière de combiner les deux méthodes. Les méthodes hybrides ont été développées dans le logiciel Minamo. L'algorithme génétique assisté par modèles RBF (Radial Basis Functions) de ce logiciel intervient dans les algorithmes hybrides en tant que méthode globale. Nous avons testé le couplage de l'algorithme génétique avec quatre méthodes locales, de type région de confiance. Deux méthodes ont été implémentées dans le cadre de ce travail tandis que des logiciels existant ont été utilisés pour les deux autres méthodes locales.

Afin de comparer les performances des méthodes hybrides, entre elles mais aussi par rapport à l'algorithme génétique assisté par modèles RBF appliqué seul, des tests ont été effectués sur un ensemble de fonctions mathématiques. L'efficacité des méthodes hybrides a ainsi pu être démontrée, d'un point de vue de la qualité de la solution obtenue mais aussi au niveau du temps d'exécution des algorithmes. Nous avons également constaté que, pour des fonctions objectif de types différents, ce sont des méthodes hybrides différentes qui se sont révélées plus performantes.

Abstract

The global optimization of expensive functions is the field in which this work is carried out. The industry is highly interested in solving such problems because it needs to handle more and more with functions of this kind. They are the result of high fidelity computer experiments. This master's thesis was carried out in partnership with the research center Cenaero.

The purpose of this master's thesis is to apply hybrid algorithms to optimize the computationnally expensive functions. Hybrid algorithms combine a global search method, such as a genetic algorithm, with a local search method. There are many ways to build hybrid algorithms due to the choice of the local method used in the coupling and also because of the way of combining the two methods. Hybrid methods have been developped in the software Minamo. The surrogate-assisted genetic algorithm (using RBF surrogates) of this software takes part in our hybrid methods as global method. We have tested the coupling of the genetic algorithm with four local methods which belong to the trust-region family. We implemented two of these local method in the framework of this work while existing softwares have been used for the two other ones.

To compare the performance of the hybrid methods, against each other but also compared with the surrogate-assisted genetic algorithm used alone, some tests on mathematical functions were performed. The hybrid methods' effectiveness has been proved, as far as the quality of the solution is concerned but also about the execution time of the algorithms. We also remark that for objective function of different kind, different hybrid methods were the most efficient.

Table des matières

Table des figures

Liste des tableaux

Introduction

Ce travail s'inscrit dans le cadre de l'optimisation, et plus précisément dans le domaine de l'optimisation numérique. Celle-ci développe des méthodes numériques, sous forme d'algorithmes, qu'on applique afin de résoudre des problèmes concrets. Nous avons réalisé ce mémoire en collaboration avec le centre de recherche Cenaero.

Le développement de nos algorithmes sera orienté de façon à pouvoir les appliquer pour résoudre le type de problèmes rencontrés par Cenaero. L'entreprise traite des problèmes en rapport avec le monde de l'industrie, ce qui implique l'utilisation de simulations haute fidélité réalisées sur ordinateur. De tels calculs peuvent devenir coûteux en temps. C'est pourquoi les méthodes développées pour résoudre de tels problèmes doivent tenir compte de ce coût et tenter de limiter l'usage de ces simulations autant que possible.

Actuellement, avec l'évolution des moyens informatiques, de tels problèmes deviennent de plus en plus fréquents à traiter. Ce mémoire s'inscrit dans le cadre de la recherche de nouvelles méthodes permettant de traiter ces problèmes. En pratique, l'utilisation d'algorithmes génétiques pour déterminer une solution de ces problèmes produit de bons résultats. Mais les algorithmes génétiques ont aussi des points faibles, notamment au niveau de la précision de la solution obtenue. La composante aléatoire de l'algorithme génétique rend difficile l'obtention d'une solution exacte, de façon (relativement) rapide. Une façon de réduire le temps nécessaire à l'exécution de l'algorithme génétique sur la fonction objectif est d'appliquer celui-ci à un modèle qui approxime la fonction exacte. On parle alors d'optimisation accélérée par modèles. D'autre part, il existe des méthodes de recherche locale qui s'avèrent être efficaces lorsqu'elles sont initiées dans une zone proche de l'optimum du problème. L'idée que nous développons dans ce travail est de construire une méthode qui couple ces deux types d'algorithmes afin d'obtenir une méthode qui combinerait au mieux les avantages des algorithmes génétiques (assistés par modèles) et des méthodes locales. De telles méthodes portent le nom d'algorithmes hybrides ou mémétiques.

Le mémoire se compose de cinq chapitres. Dans le Chapitre 1, nous introduisons des notions de base d'optimisation qui nous seront utiles pour la suite du travail. Ensuite, le Chapitre 2 présente l'entreprise Cenaero avec laquelle nous réalisons ce mémoire, mais aussi le logiciel Minamo développé par cette entreprise. C'est au coeur de ce logiciel que nous avons implémenté nos méthodes locales afin de construire l'algorithme hybride. Ce dernier fait également intervenir l'algorithme génétique et les modèles RBF (Radial Basis Fonctions) déjà implémentés dans Minamo. Dans le Chapitre 3, nous

synthétisons dans un premier temps nos investigations sur le couplage d'un algorithme génétique et d'une méthode locale que nous avons réalisées dans la littérature. Sur base des différents éléments que nous y avons découverts, nous présentons ensuite le schéma que nous avons choisi pour nos méthodes hybrides. La description des méthodes locales qui interviennent dans le couplage fait l'objet du Chapitre 4. Toutes sont des méthodes de type région de confiance. Nous décrivons en détails les deux méthodes que nous avons implémentées. Nous présentons également les deux autres méthodes pour lesquelles nous avons utilisé des logiciels déjà existants. Finalement, le Chapitre 5 regroupe les éléments de notre travail qui se rapportent aux tests numériques effectués. Nous commençons par la présentation des fonctions mathématiques utilisées pour les tests. Nous discutons et fixons ensuite les nombreux paramètres qui interviennent tout au long de l'algorithme hybride, avant de présenter les résultats de nos tests. Nous terminons ce travail par quelques conclusions et perspectives.

Chapitre 1

Notions de base en optimisation

Ce premier chapitre a pour but d'introduire les notions d'optimisation que nous utiliserons dans la suite de ce mémoire. Dans ce travail, nous allons étudier le couplage entre un algorithme génétique et des méthodes de recherche locale dans l'optique d'améliorer les performances de l'algorithme génétique. Les rappels s'articuleront donc autour de ces deux domaines.

Dans la première section, nous commencerons par rappeler dans quels contextes on utilise l'adjectif "local" mais aussi celui qui lui est opposé : "global". Nous parlerons également de conditions nécessaires d'optimalité. La seconde section sera composée d'une description non exhaustive de méthodes d'optimisation. Les méthodes locales seront d'abord présentées avant de s'intéresser aux méthodes globales. La dernière section sera quant à elle entièrement consacrée aux algorithmes génétiques. Leur fonctionnement ainsi que leurs principales caractéristiques seront présentés. Nous nous centrerons plus particulièrement sur le type d'algorithme qui interviendra dans le couplage développé dans le Chapitre 3.

1.1 Caractéristiques locales, globales

Les termes "local" et "global" sont souvent utilisés en optimisation. Ils peuvent être employés dans deux contextes différents : d'une part lorsqu'on parle du type de solution optimale, d'autre part quand on traite du type de convergence d'une méthode. Il est important de ne pas confondre ces deux contextes d'utilisation. Une fois la distinction marquée, nous présentons brièvement quelques méthodes d'optimisation, aussi bien locales que globales.

Avant de développer ces différents aspects, nous allons préciser le problème que nous considérerons dans cette section. Il s'agit d'un problème de *minimisation sans contrainte* qui peut s'exprimer sous la forme

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1.1}$$

où x est un vecteur à n composantes réelles et $f : \mathbb{R}^n \to \mathbb{R}$ est une fonction à valeurs réelles deux fois continûment différentiable.

1.1.1 Définition d'une solution optimale

En raison de notre hypothèse de travail (1.1), une solution optimale est un minimum, qui peut être de deux types : global ou local.

Définition 1.1. (Minimum global)
Un point $x^* \in \mathbb{R}^n$ *est un* minimum global *de* f *si* $f(x^*) \leq f(x)$ *pour tout* $x \in \mathbb{R}^n$.

Définition 1.2. (Minimum local)
Un point $x^* \in \mathbb{R}^n$ *est un* minimum local *de* f *s'il existe un voisinage* $\mathcal{V}$ *de* x^* *tel que* $f(x^*) \leq f(x)$ *pour tout* $x \in \mathcal{V}$.

La Figure 1.1[1] illustre ces définitions issues de [11]. La fonction représentée possède un minimum global et deux minima locaux.

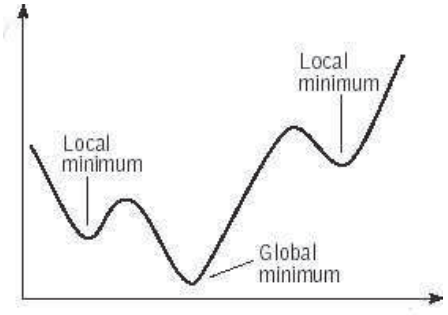


FIGURE 1.1 – Représentation de minima locaux et globaux.

Le minimum global, présenté à la Définition 1.1, est très souvent recherché dans les applications industrielles. Cependant, une telle solution est généralement difficile à déterminer car l'information sur la fonction dont on dispose est locale et le problème peut posséder de nombreux minima locaux. La plupart des méthodes fournissent donc un minimum local comme résultat. Un cas particulier est à signaler : celui des fonctions convexes. Pour ces problèmes, la distinction entre les 2 types de minimum n'a plus lieu d'être. Tout minimum local de f est en effet un minimum global de f. De plus, si f est strictement convexe, le minimum est unique (cfr. [2]). Toutefois, dans les applications réelles, on traite souvent des fonctions dont on ne connaît pas l'expression analytique car elles sont calculées via un outil de simulation par exemple. Sans l'expression analytique, il est impossible de détecter la convexité de la fonction. On ne pourra donc pas l'exploiter, le cas échéant, pour assurer le caractère global de la solution obtenue.

1.1.2 Conditions nécessaires d'optimalité

Nous présentons maintenant un moyen de détecter des solutions potentielles. En effet, tout minimum doit vérifier la condition nécessaire d'optimalité du premier ordre, présentée par le Théorème 1.1 (extrait de [2]). Cette

1. `http://science.jrank.org/pages/47754/inverse-methods.html`, consultation le 3 mai 2010.

condition est importante car elle est souvent utilisée en tant que critère d'arrêt par les algorithmes d'optimisation locale, comme nous pourrons le constater ultérieurement.

Théorème 1.1. (Condition nécessaire d'optimalité du premier ordre) *Soit x^* un minimum local d'une fonction $f : \mathbb{R}^n \to \mathbb{R}$. Si f est différentiable dans un voisinage ouvert $\mathcal{V}$ de x^*, alors,*

$$\nabla f(x^*) = 0,$$

où $\nabla f(x^*)$ représente le gradient de f en x^*.

Les points qui vérifient la condition du Théorème 1.1 sont appelés *points stationnaires*. Les minima, mais aussi les maxima et les points de selle, sont des points stationnaires. Il existe des conditions du second ordre (nécessaire et suffisante) qui permettent de distinguer les minima parmi les points de selle et les maxima. Nous ne nous attardons pas sur cette dernière condition car nous ne l'utiliserons pas dans la suite de nos développements. Ce sujet est traité par Bierlaire dans [2].

1.1.3 Caractérisation des types de convergence

La plupart du temps, les méthodes d'optimisation suivent des schémas itératifs qui produisent une séquence d'itérés successifs. Notons cette suite $\{x_k\}$. Les algorithmes nécessitent souvent une infinité d'itérations pour atteindre une solution exacte. On parle donc de la convergence de la suite $\{x_k\}$ vers la solution exacte x^*. La convergence globale est définie par Bierlaire dans [2] de la façon suivante.

Définition 1.3. (Convergence globale)
Soit un algorithme itératif qui génère une suite $\{x_k\}$ dans $\mathbb{R}^n$ afin de résoudre le problème de minimisation sans contrainte (1.1). L'algorithme est dit globalement convergent *si*

$$\lim_{k\to\infty} ||\nabla f(x_k)|| = 0,$$

quel que soit le point de départ $x_0 \in \mathbb{R}^n$.

La notation $||.||$ désigne une norme vectorielle sur $\mathbb{R}^n$. La Définition 1.3 implique que, lorsqu'un algorithme est globalement convergent, on est assuré de converger vers un point stationnaire *quel que soit le point de départ x_0 choisi*. Ce point, s'il est un minimum, sera une solution locale mais pas forcément globale. Par opposition, une méthode sera qualifiée de *localement convergente* si le point initial doit être choisi proche d'un optimum (i.e., une solution) pour qu'il y ait convergence de la méthode.

La distinction entre *solution globale* et *convergence globale* apparaît maintenant plus clairement. Comme on vient de le préciser, il existe des algorithmes qui peuvent converger de façon globale mais vers un minimum local. C'est le cas par exemple des méthodes de région de confiance que nous décrirons plus amplement dans la suite de ce travail.

1.2 Méthodes d'optimisation

Nous allons maintenant présenter succinctement quelques méthodes classiques d'optimisation. Le but est de montrer qu'il en existe plus d'une et que celles que nous étudierons lors du couplage sont loin d'être l'unique choix possible. Celles que nous qualifierons de "locales" le seront en raison de leur convergence vers une solution locale, quel que soit le type de cette convergence. D'autre part, les méthodes permettant de déterminer une solution globale seront dites "globales". Les ouvrages [2, 3, 5, 8, 9, 11] ont été utilisés afin de réalisé cette section. Pour les méthodes globales, nous avons également consulté [12].

1.2.1 Les méthodes locales

Nous allons considérer 3 classes de méthodes locales. La première est celle qui regroupe des algorithmes qui suivent un schéma du type "recherche linéaire". Nous décrivons ensuite les méthodes de "région de confiance" avant de terminer avec des méthodes d'optimisation "sans dérivées".

Les méthodes de recherche linéaire

Commençons par introduire le schéma général d'une méthode de recherche linéaire. Celui-ci est présenté par l'Algorithme 1.1.

Algorithme 1.1. (Algorithme général pour une recherche linéaire)

1. Initialisation : donner un point initial x_0 et poser $k = 0$.
2. Itérations :
   a. Déterminer une direction de descente d_k (i.e., tq. $\nabla f(x_k)^T d_k < 0$),
   b. Déterminer une longueur de pas α_k,
   c. Mettre à jour : $x_{k+1} = x_k + \alpha_k d_k$,
   d. Incrémenter : $k = k + 1$.
3. Critère d'arrêt : $||\nabla f(x_k)|| \approx 0$.

Suivant le choix de calcul pour déterminer la direction de descente et la longueur du pas, on obtient différentes méthodes. La première et la plus simple est celle dite *de la plus forte pente*. La direction de descente dans ce cas est définie par l'équation

$$d_k = -\nabla f(x_k). \tag{1.2}$$

Le nom de cette méthode provient du fait que la direction de l'opposé du gradient est en fait celle dans laquelle la fonction décroît localement le plus fortement à partir de x_k. Pour le calcul de la longueur de pas α_k, on utilise une recherche linéaire. L'idée consiste à trouver la distance à parcourir pour minimiser la fonction lorsqu'on se déplace le long de la direction de descente donnée par (1.2). Cela revient à trouver la solution α_k du problème

$$\min_{\alpha>0} f(x_k + \alpha d_k). \tag{1.3}$$

L'avantage de la méthode de la plus forte pente est qu'elle ne nécessite pas le calcul de dérivées secondes. Cependant, lorsque la recherche linéaire est exacte, i.e., α_k est le minimum exact de (1.3), on peut montrer que deux directions de descente successives sont orthogonales [14]. Cela provoque un comportement "en zigzag" de la suite $\{x_k\}$, ce qui implique une certaine lenteur de la méthode. Ce phénomène est illustré par la Figure 1.2. [2]

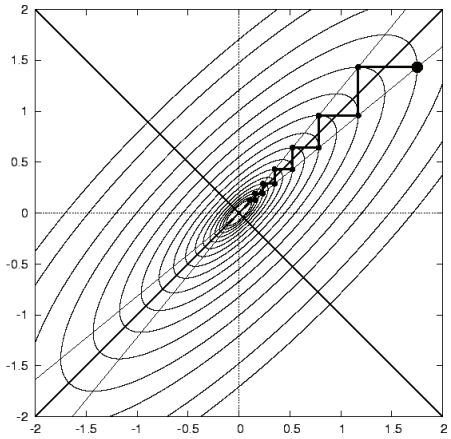


FIGURE 1.2 – Comportement en zigzag de la suite des itérés pour la méthode de la plus forte pente.

La seconde méthode est celle de *Newton* sous sa forme classique. La direction de descente [3] d_k est la solution des équations de Newton

$$\nabla^2 f(x_k)\ d = -\nabla f(x_k) \tag{1.4}$$

et la longueur de pas α_k est fixée à 1 pour toute valeur de k. L'étape de mise à jour de l'Algorithme 1.1 peut donc se réécrire [4] sous la forme

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k).$$

En pratique, on évitera de calculer l'inverse de la matrice hessienne et on préférera résoudre le système d'équations (1.4).

La méthode de Newton peut aussi être vue comme la minimisation d'un modèle quadratique (convexe) qui approxime la fonction f. L'approximation s'obtient en considérant une série de Taylor au second ordre :

$$f(x_k + d) \approx f(x_k) + \langle \nabla f(x_k), d \rangle + \frac{1}{2} \langle d, \nabla^2 f(x_k) d \rangle, \tag{1.5}$$

où $\langle ., . \rangle$ représente le produit scalaire sur $\mathbb{R}^n$. Minimiser cette approximation par rapport à la direction de descente d revient, par application de la condition nécessaire d'optimalité, à résoudre l'équation

$$\nabla f(x_k) + \nabla^2 f(x_k) d = 0$$

2. `http://komarix.org/ac/papers/thesis/thesishtml/node\-10.html`, consultation le 3 mai 2010.

3. Si la matrice hessienne calculée à l'itéré courrant $\nabla^2 f(x_k)$ est définie positive, d_k est une direction de descente, cfr. [2].

4. Dans le cas où la matrice hessienne est inversible.

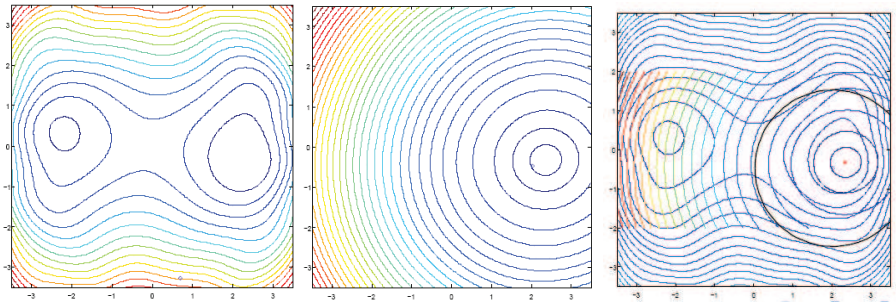
qui est équivalente à l'équation (1.4).

Le grand avantage de la méthode de Newton est sa convergence rapide. On peut prouver que si le gradient de la fonction objectif est Lipschitz continu[5], la suite $\{x_k\}$ converge de façon quadratique vers la solution x^*. Cependant, la convergence n'est assurée que localement. Pour obtenir la convergence globale, on ajoute une recherche linéaire pour déterminer la longueur de pas α_k, comme on l'a fait dans la méthode de la plus forte pente. Un autre inconvénient de cette méthode est qu'elle nécessite le calcul des dérivées secondes. Celui-ci peut se révéler coûteux, ce qui réduit l'efficacité de la méthode. Les méthodes *quasi-Newton* permettent d'éviter le calcul de la matrice hessienne exacte en l'approchant. Plusieurs types de formules peuvent être utilisées pour l'obtenir, notamment le calcul par différences finies ou la mise à jour BFGS. Ces méthodes sont décrites plus amplement dans [2] et [11].

Les méthodes de région de confiance

L'utilisation d'une région de confiance est une technique de "globalisation". Comme les recherches linéaires, les régions de confiance permettent de garantir la convergence d'un algorithme à partir de n'importe quel point de départ. Ces méthodes sont décrites par Conn, Gould et Toint dans [5].

On travaille avec un modèle (souvent quadratique) de la fonction objectif. La principale différence par rapport aux méthodes précédentes vient du fait qu'on définit une région autour du point x_k dans laquelle on suppose que le modèle est suffisamment proche de la vraie fonction. Cette zone porte le nom de *région de confiance*. La Figure 1.3 représente les courbes de niveau d'une fonction, les courbes de niveau de son modèle quadratique autour du point x_k et la superposition des 2 familles de courbes de niveau sur lesquelles on a aussi tracé la région de confiance autour de x_k. On observe qu'à l'intérieur de la région, le modèle est relativement proche de la fonction (source : [15]).



(a) la fonction (b) le modèle (c) supperposition de la fonction et du modèle

FIGURE 1.3 – Représentation des régions de confiance.

Le schéma de la méthode des régions de confiance est légèrement différent de celui des méthodes de recherche linéaire. Le calcul de la direction de

5. Une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est dite *Lipschitz continue* s'il existe une constante $M > 0$ telle que pour tout $x, y \in \mathbb{R}^n$ $||f(x) - f(y)|| \leq M||x - y||$.

descente d_k et de la longueur de pas α_k se fait dans une même étape lors du calcul de la quantité s_k, qu'on appellera le pas, mais le nouveau point candidat ainsi obtenu ne sera plus nécessairement accepté comme c'était le cas pour les méthodes de recherche linéaire.

L'algorithme consiste donc d'abord à calculer un pas s_k qui est déterminé de façon à ce que $x_k + s_k$ appartienne à la région de confiance et qu'il réduise suffisamment la valeur du modèle. On compare ensuite cette diminution à la variation réelle produite dans la fonction f. Si la prédiction obtenue par le modèle se révèle suffisante, on peut accepter le nouvel itéré défini par

$$x_{k+1} = x_k + s_k.$$

Sinon, on restreint la région de confiance afin d'avoir un modèle plus proche de f et on calcule à nouveau s_k. On réduit le rayon jusqu'à accepter un nouvel itéré. Par contre, dans le cas où le candidat est accepté et le modèle suffisamment bon, on augmente le rayon de la région de confiance. En itérant, on obtient une suite $\{x_k\}$ dont on peut prouver la convergence globale vers une solution optimale. Les preuves de convergence peuvent être consultées dans [5], Chapitre 6. Nous reparlerons plus en détails des régions de confiance dans la suite de ce travail (cfr. Chapitre 4), puisque c'est une méthode que nous tenterons d'employer dans l'application du couplage.

Les méthodes d'optimisation sans dérivée

Le dernier type d'algorithme que nous présenterons ici est celui des méthodes *DFO* (pour *Derivative Free Optimization*), [6]. Comme l'indique leur nom, celles-ci peuvent s'appliquer à des fonctions dont les dérivées existent mais on ne sait pas (ou ne veut pas) calculer celles-ci. Ces méthodes sont dites d'ordre 0 car elles n'utilisent que l'expression de la fonction objectif et aucune dérivée. Il existe différentes méthodes qui ne font pas appel aux dérivées, citons entre-autres celles qui n'utilisent pas la fonction objectif mais un modèle qui l'approche, la méthode de Nelder et Mead qui utilise le simplexe ou encore des méthodes de directions conjuguées. Nous ne présenterons ci-après que la première méthode citée. Nous la décrirons plus longuement dans la suite de ce travail car nous l'utiliserons dans notre méthode hybride. Son déroulement ne peut pas être comparé à l'Algorithme 1.1. On peut par contre le diviser en deux étapes : on crée un modèle de la fonction à optimiser et on le minimise ensuite dans une région de confiance. Il s'agit d'une structure similaire à celle des méthodes de région de confiance.

La définition du modèle ne peut plus se faire par une approximation via une série de Taylor (comme l'équation (1.5) dans la méthode de base des régions de confiance) puisqu'on ne dispose pas des dérivées de la fonction objectif. Le modèle peut, par contre, être construit par une interpolation polynomiale à partir d'un certain nombre de points où la valeur de la fonction objectif a été évaluée. Lorsqu'on effectuera la construction de ce modèle, il faudra être vigilant et réaliser un choix judicieux des points d'interpolation pour permettre une bonne correspondance entre le modèle et la fonction. En effet, des points "mal disposés" dans l'espace peuvent fournir une approximation inappropriée. Le résultat fourni par la minimisation dans la région

de confiance pourra être évalué via la vraie fonction objectif. Ceci permettra d'enrichir l'ensemble de points disponibles pour l'interpolation. Le modèle construit à l'itération suivante pourra alors éventuellement être de meilleure qualité. Cette technique sera développée plus amplement au Chapitre 4.

1.2.2 Les méthodes globales

Les méthodes globales sont plus compliquées que les méthodes locales dans le sens où on ne peut plus uniquement appliquer les conditions classiques d'optimalité. Le critère d'annulation du gradient ne suffit plus pour déterminer si on a atteint la solution souhaitée puisqu'il ne permet pas d'assurer que la solution trouvée soit bien un minimum *global*.

Les méthodes disponibles sont nombreuses et leur classification n'est pas évidente car elle peut se baser sur différents critères. Un premier est le type de recherche effectuée. Elle peut se faire au hasard, dans un voisinage proche de l'itéré courant ou encore dans tout l'espace et permettre de "sauter" d'un minimum local à un autre. Une seconde distinction peut être le caractère déterministe ou stochastique de la méthode. Ces deux classifications peuvent se "cumuler", une méthode peut posséder des caractéristiques provenant des 2 classifications. On peut ajouter un troisième critère qui distingue les méthodes qui garantissent la convergence vers une solution globale de celles qui ne sont basées que sur des heuristiques. On pourrait encore en envisager d'autres.

Ces différentes caractéristiques peuvent également être combinées pour améliorer la qualité de l'algorithme. Si on prend le premier critère par exemple, appliquer une exploration dans tout l'espace dans un premier temps permet d'éviter d'ignorer la zone du minimum global tandis qu'une recherche plus localisée sur la fin permettra de se rapprocher du minimum. On retrouve cette démarche, par exemple, dans l'algorithme du recuit-simulé, [13]. Le paramètre principal de cet algorithme est une valeur de température qui décroît progressivement au plus le nombre d'itérations augmente. Au plus la température est élevée, au plus la probabilité d'accepter le nouveau candidat choisi aléatoirement est grande, ce qui permet une exploration de l'espace des solutions. Puis, quand la température faiblit, les points acceptés restent de plus en plus localisés dans la même région. A la fin, la recherche est donc plus locale et permet de déterminer l'optimum dans la zone courante. Le but de la phase d'"exploration" est d'éviter de rester piégé dans un minimum local et de tenter de déterminer l'optimum global.

Nous pouvons constater que cette dernière méthode ne fait pas intervenir les dérivées. Il en existe un autre groupe qui ne nécessite pas leur calcul et permet aussi de déterminer un optimum global. Il s'agit des *algorithmes génétiques* qui font partie de la famille des *algorithmes évolutionnaires*. On peut également les classer dans la famille des méthodes stochastiques car ils font intervenir de l'aléatoire. Nous allons les décrire dans la section qui suit car ils interviendront dans le couplage que nous étudierons dans la suite de ce travail. Il existe encore d'autres méthodes globales, comme l'algorithme de colonies de fourmis ou l'optimisation par essaims de particules par exemple. Nous ne développons pas ces méthodes plus amplement puisque la méthode

globale qui nous intéresse est l'algorithme génétique.

1.3 Algorithmes génétiques

Les algorithmes génétiques font partie de la famille des algorithmes évolutionnaires. Leur origine remonte aux années 1960. John Holland est souvent considéré comme leur fondateur. David Goldberg, un de ses étudiants, a par la suite continué et approfondi son travail. Comme l'indique leur nom, ces algorithmes empruntent des concepts à la biologie tels que l'évolution d'une population, le génotype, les mutations, la sélection, etc.

Les algorithmes génétiques possèdent différents avantages. Haupt [10] en présente quelques-uns. Parmi ceux-ci, on peut citer le fait qu'ils ne nécessitent pas les dérivées, qu'ils permettent l'optimisation avec des variables continues ou discrètes, qu'ils peuvent s'échapper d'une zone d'un minimum local ou encore qu'ils peuvent s'appliquer aussi bien à des fonctions analytiques qu'à des données numériques ou obtenues par expériences. Les algorithmes génétiques sont également applicables à des fonctions multimodales (i.e., qui présentent plusieurs optima), bruitées ou encore fortement non linéaires. Ils permettent aussi de résoudre des problèmes d'optimisation multi-objectifs. Ces caractéristiques expliquent pourquoi ce type de méthodes peut être utilisé pour des applications industrielles. On pourra d'une part éviter le plus possible les optima locaux et, d'autre part, la méthode reste compatible avec des outils de simulation numérique, souvent utilisés dans les applications industrielles.

Dans la suite de cette section, nous commencerons par expliquer le fonctionnement général des algorithmes génétiques. Nous décrirons ensuite quelques formes de codage des données ainsi que quelques opérateurs et processus de sélection qui interviennent dans le fonctionnement de ces algorithmes. Nous nous basons sur [1], [10], [13], [33] et [34].

1.3.1 Principes généraux

Un algorithme génétique fait évoluer une population d'individus au cours du temps. Ces individus représentent des solutions potentielles. Les transformations de la population se font suivant les lois de l'évolution naturelle. Au cours du temps, on va ainsi ne conserver que les meilleurs candidats. La population initiale est souvent générée aléatoirement. Sa taille reste généralement fixe au cours du temps.

Pour déterminer quels sont les meilleurs individus à garder dans la population, on a besoin d'une mesure. On l'appelle la *viabilité* ou la *fonction de fitness* de l'individu. Elle se base sur l'objectif à atteindre dans le problème et sur les caractéristiques de l'individu. Celles-ci sont représentées dans les *gènes*. Un individu se compose donc d'un ensemble de gènes, nommé le *génotype*. Les valeurs que peuvent prendre ces derniers sont appelées *allèles*. Par exemple, un individu codé sous forme binaire et de longueur 10 pourra être réprésenté par le génotype suivant

$$1001101000,$$

où les allèles sont 1 et 0.

Le schéma général de l'algorithme génétique peut être divisé en 4 étapes dont les principales sont la *sélection* et la *variation*. Toutes ces étapes sont reprises dans l'Algorithme 1.2 [13]. On note $P(t)$ la population au temps t. Elle est de taille m (fixée).

Algorithme 1.2. (Algorithme génétique)

1. INITIALISATION : Poser $t = 0$ et générer une population initiale $P(0)$ avec m individus créés aléatoirement.
2. SÉLECTION : Générer $P'(t)$ en modifiant les individus de $P(t)$ proportionnellement à leur fitness.
3. VARIATION : Générer $P(t+1)$ en appliquant les opérateurs génétiques aux individus de $P'(t)$.
4. INCRÉMENTATION : Poser $t = t + 1$ et retourner à l'étape 2.

Durant l'étape de sélection, on modifie la population en utilisant la fonction de fitness comme référence. Les individus qui présentent une fitness élevée vont rester présents dans la nouvelle population tandis que les mauvais candidats seront quant à eux représentés en quantité plus faible voire même supprimés de cette nouvelle population.

L'étape de variation a pour but de faire évoluer la population en changeant les caractéristiques des individus (et donc les valeurs prises par leurs gènes). Ces modifications résultent de l'application d'*opérateurs génétiques* comme l'opérateur de croisement et de mutation.

1.3.2 Codage des données

Dans l'exemple de génotype que nous avons donné ci-dessus, nous avons supposé qu'il s'agissait d'un individu codé sous forme binaire. Le *codage binaire* est celui qui a été utilisé dès le début du développement des algorithmes génétiques. Un individu est donc représenté par une liste de 0 et de 1. C'est la représentation la plus simple et qui est souvent utilisée pour la description des opérateurs. Pour certains problèmes, cette forme de codage peut se révéler inadaptée. C'est le cas par exemple des problèmes de grande taille[6] qui demandent une certaine précision numérique. La représentation binaire nécessiterait des gènes de trop grande taille et on ne pourrait pas nécessairement obtenir une précision suffisante. On adopte alors le *codage réel*. On peut alors parler d'algorithme génétique *continu*.

Un individu est alors représenté par un vecteur d'éléments réels, codés en virgule flottante. La longueur du vecteur correspond à la dimension du problème. Dans le cas d'un problème à n variables qui prennent respectivement les valeurs v_{ji} pour l'individu j $(i = 1, ..., n)$, le génotype de l'individu j sera le vecteur à n composantes

$$(v_{j1}\ v_{j2}\ ...\ v_{jn})$$

6. C'est-à-dire où le génotype est fort long, ce qui peut être dû à un nombre important de variables dans le problème.

et la population de taille m pourra être représentée par la matrice $m \times n$

$$P = \begin{pmatrix} v_{11} & v_{12} & \dots & v_{1n} \\ v_{21} & v_{22} & \dots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \dots & v_{mn} \end{pmatrix},$$

où chaque ligne de la matrice P représente un individu. Avec cette représentation, la fonction de fitness est très facile à calculer. Il suffit d'évaluer la fonction objectif du problème pour les valeurs des variables qui constituent le génotype.

Ce type de codage est celui qui est employé dans l'algorithme génétique qui interviendra dans le couplage avec les méthodes de recherche locale que nous effectuerons plus tard. Nous allons donc concentrer les descriptions qui suivent pour des données encodées sous forme réelle.

1.3.3 Processus de sélection

Le but des processus de sélection est de créer une nouvelle population en suivant les lois de l'évolution naturelle. Les meilleurs individus seront conservés et multipliés tandis que les moins bons seront présents en plus faible quantité. Il existe plusieurs façons d'effectuer cette sélection et nous n'en présenterons ici que quelques-unes d'entre elles.

La méthode la plus classique est la sélection *"roulette wheel"*. Il s'agit de sélectionner les individus proportionnellement à leur fitness. On commence par attribuer à chaque individu j un sous-intervalle de l'intervalle [0,1] dont la longueur $L(j)$ est définie par

$$L(j) = \frac{F(j)}{\sum_{k=1}^{m} F(k)},$$

où $F(j)$ désigne la fitness de l'individu j et m la taille de la population. En juxtaposant ces m intervalles, on peut recouvrir [0,1] de façon à ce que chaque individu occupe une place proportionnelle à sa fitness. La longueur de chaque intervalle représente la probabilité du choix de l'individu qui lui correspond. Cette construction est illustrée à la Figure 1.4. Il faut ensuite sélectionner

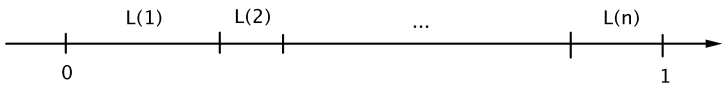


FIGURE 1.4 – Intervalles représentant la probabilité de choix de chaque individu

les individus qui feront partie de la nouvelle population. Si on suppose qu'ils sont au nombre de l, on génère l nombres aléatoires entre 0 et 1, de façon uniforme. Ce sont les individus dont les intervalles contiennent ces l valeurs qui sont retenus. Les individus dont la fitness est la meilleure occuperont

un intervalle de longueur plus grande et auront donc plus de chances d'être sélectionnés pour appartenir à la nouvelle population. Un inconvénient de ce mode de sélection est qu'il est sensible à la variance de la fitness. Une trop grande variance va conduire à une perte rapide de beaucoup d'individus, ce qui va empêcher de garder une diversité suffisante parmi les individus. Au contraire, une petite variance va conduire à une sélection proche d'un tirage au sort puisque tous les individus possèdent une probabilité de choix fort semblable.

Pour remédier à ce problème, on peut appliquer le processus de sélection *sigma-scaling*. Il consiste à appliquer le processus roulette wheel après avoir modifié la fitness. La modification est une sorte de normalisation. On soustrait de chaque fitness la fitness moyenne puis on divise chacune par l'écart type. On termine en ajoutant à chaque fitness la valeur de la plus petite d'entre elles. On obtient ainsi des valeurs positives et bien distribuées.

Une autre méthode de sélection est celle appelée le *tournoi*. Le principe est simple. On confronte deux individus choisis au hasard. Celui qui possède la meilleure fitness obtient une probabilité $p > 0.5$ d'être sélectionné tandis que l'autre reçoit une probabilité de $1 - p$. Pour sélectionner l individus, on répète ce processus l fois.

L'élitisme est aussi une méthode qui fait partie des processus de sélection. Ce mode de sélection permet, comme l'indique son nom, de conserver les meilleurs individus. On leur réserve un nombre x de places dans la nouvelle population qu'on remplit en choisissant parmi les y meilleurs individus grâce à des méthodes décrites précédemment. Celles-ci sont également utilisées pour compléter le reste de la nouvelle population. On peut aussi apporter des variantes à l'élitisme en jouant sur les valeurs des paramètres x et y : il peuvent être égaux (on garde une fois chaque meilleur individu), $x > y$ (on sélectionne plusieurs fois certains individus), etc.

1.3.4 Opérateurs génétiques

Nous terminons ce chapitre en décrivant le *croisement* et la *mutation* qui sont les 2 opérateurs génétiques principaux.

Le croisement, qu'on nomme aussi *cross-over*, est l'opérateur le plus utilisé. A partir de deux individus (les parents), il en crée deux nouveaux (les enfants). Il existe de nombreuses variantes de la méthode de cross-over. La plus basique est celle qui consiste à déterminer un ou plusieurs points de coupure du génotype, de façon aléatoire. Les petits groupes de variables créés par les coupures sont ensuite permutés pour créer les enfants. La Figure 1.5 illustre le processus pour un point de coupure tandis que la Figure 1.6 le représente lorsqu'il y a deux points. Si on choisit autant de points de coupure qu'il y a de variables, on peut obtenir un cross-over dit *uniforme*. Pour chaque élément qui compose le gène d'un enfant, on décide via un tirage aléatoire s'il hérite de la valeur de son parent numéro 1 ou de son parent numéro 2.

Un autre type de cross-over est celui qui consiste à combiner des éléments de chaque parent pour créer les enfants. La valeur de la $k^{\text{ième}}$ composante d'un enfant, notée v_k, peut par exemple être obtenue par une équation du

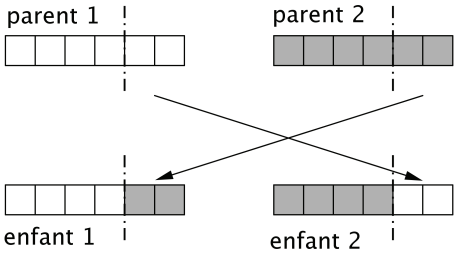


FIGURE 1.5 – Cross-over à un point.

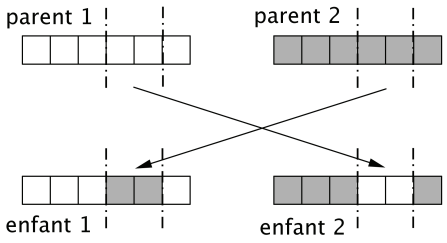


FIGURE 1.6 – Cross-over à 2 points.

type

$$v_k = \alpha p_{1k} + (1 - \alpha) p_{2k}$$

où $\alpha \in [0, 1]$, p_{1k} représente la valeur de la k^{ieme} composante du parent 1 et p_{2k} celle du parent 2. L'avantage de telles combinaisons est qu'elles permettent d'amener des nouvelles valeurs dans les gènes de la population. Les premières techniques se contentaient de permuter des valeurs déjà présentes dans un individu, ce qui peut se révéler insuffisant pour parcourir l'espace puisque les variables prennent des valeurs continues. Il est également possible de combiner différents types de cross-over. On peut par exemple déterminer une variable comme point de coupure. La valeur de cette variable chez les enfants sera calculée par combinaison des valeurs des parents tandis que les variables situées à gauche du point de coupure resteront inchangées et celles se trouvant à droite de ce point seront permutées comme pour le cross-over à un point illustré à la Figure 1.5.

Le second opérateur, la mutation, est très simple à mettre en place. Il consiste à sélectionner aléatoirement un gène d'un individu (le parent). La valeur de ce gène est ensuite changée, toujours de façon aléatoire, afin d'obtenir l'enfant. La Figure 1.7 illustre cette transformation. La mutation ne doit pas être appliquée à une trop grande partie de la population. Elle a pour effet d'introduire un individu représentant une nouvelle région de l'espace de solutions, d'amener de nouvelles valeurs. Cela s'avère donc bénéfique pour la phase d'exploration de l'espace des solutions afin de s'échapper d'une zone de minimum local, mais si on applique trop souvent cet opérateur, le risque

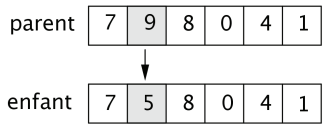


FIGURE 1.7 – Mutation du deuxième gène.

est de perdre la qualité de la solution déjà obtenue.

Une autre technique de mutation consiste à ajouter un "bruit" à la valeur actuelle de la variable sélectionnée pour être mutée. Il suffit d'ajouter un nombre aléatoire généré suivant une distribution normale de moyenne 0 et de variance σ. Il faut dans ce cas bien choisir le paramètre σ.

Chapitre 2

Présentation de Cenaero et du logiciel Minamo

Ce chapitre se divise en deux parties. Nous commençons par présenter l'entreprise Cenaero avec laquelle nous avons réalisé ce mémoire. La seconde section est consacrée à la description du cadre de travail, et plus particulièrement au logiciel Minamo qui constitue la base de notre couplage. Nous décrivons le principe général de fonctionnement de Minamo mais aussi les différentes étapes qui le composent.

2.1 L'entreprise Cenaero

Cenaero est un centre de recherches appliquées, créé en mars 2002 et situé dans l'aéropôle de Gosselies (Belgique). Le centre a été fondé par des actionnaires venant du monde de l'industrie mais aussi des universités. Les activités de Cenaero se présentent sous deux aspects. D'une part, l'entreprise fournit des services aux industries tels que des méthodes de simulation numérique de haute qualité. Dans son rapport d'activité pour l'année 2009 [36], Cenaero expose son ambition :

> "Notre ambition est d'être reconnu sur le plan international en tant que leader des technologies de modélisation et simulation numérique, d'être un partner stratégique de grandes industries mais aussi un support pour les compagnies régionales, y compris les petites et moyennes entreprises."

D'autre part, Cenaero participe à des projets de recherche et développement à plus long terme, notamment des projets européens. L'avantage de ces deux types d'activité est de pouvoir intégrer des découvertes scientifiques récentes dans les services fournis aux entreprises, afin de répondre au mieux à leur demande.

Cenaero se présente, sur son site et dans le rapport d'activité annuel, au travers de ses valeurs. La première est la passion qui les pousse à réussir avec créativité les challenges proposés par les clients. Ensuite, l'entreprise insiste sur l'importance d'une relation de confiance à long terme avec les clients, mais aussi au sein même de l'entreprise. Pour aller de l'avant et mener à

bien d'ambitieux projets, l'audace est un élément important. Enfin, prendre soin de ses collaborateurs est un élément clé du succès de l'entreprise.

Les activités de Cenaero sont majoritairement centrées sur le domaine de l'aéronautique, qui représente 65% de ses affaires. Cette orientation se remarque également dans l'identité de ses partenaires et clients. Les noms de quelques-uns sont donnés sur le site de l'entreprise [35]. Depuis 2007, Cenaero est partenaire R&D du groupe SAFRAN. Il s'agit d'un équipementier international de haute technologie, leader en aéronautique, défense et sécurité. Snecma, le principal client de Cenaero fait partie du groupe SAFRAN. Au niveau des clients, on peut citer SABCA, SONACA, AIRBUS ou encore Techspace Aero qui fait partie du groupe SAFRAN. Parmi les clients, on retrouve aussi des sociétés de secteurs tout à fait différents de l'aéronautique telles que Renault, Bosh, IBA ou encore Cardiatis qui produit du matériel médical.

Le but de Cenaero est de s'occuper des soucis de l'industrie en offrant des services flexibles et un engagement de haut niveau. L'entreprise offre une large gamme de services, basés sur une expertise de la conception assistée par ordinateur (Computer Aided Engineering). L'équipe de Cenaero se compose principalement d'ingénieurs et de docteurs très qualifiés et spécialisés en méthodes numériques, modelage des matériaux, dynamique des fluides, physique et mathématiques appliquées. Ils sont plus de 50 à faire partie de l'entreprise. Ils se répartissent en 4 grands groupes de recherche scientifique, décrits ci-après :

Virtual Manufacturing (Fabrication virtuelle)
: Le rôle de ce groupe est de créer des simulations virtuelles de processus industriels tels que les procédures de soudure d'éléments et les traitements de chaleur. Le groupe a développé un logiciel, Morfeo, qui est orienté éléments finis.

Multi-scale Modeling of Materials and Structures (Modélisation des matériaux et des structures)
: Ce groupe travaille sur les matériaux composites, métalliques et hybrides. Il étudie leur structure, les fissures qui peuvent y apparaître, etc.

CFD Multi-Physics (Mécanique des fluides multi-physique)
: Le sujet d'étude de ce troisième groupe est la mécanique des fluides. Un exemple est l'étude des flots d'airs ou de liquide pour des applications de propulsion.

Multi-Disciplinary Optimization (Optimisation multi-disciplinaire)
: Les activités du groupe d'optimisation sont centrées autour du développement du logiciel Minamo et de son déploiement sur des applications industrielles. Il s'agit d'un logiciel d'optimisation basé sur des méta-modèles et qui utilise un algorithme génétique qui peut gérer les optimisations mono- et multi-objectifs. L'objectif d'une partie du groupe d'optimisation est de développer le logiciel en implémentant de nouveaux modèles et de nouveaux algorithmes plus performants afin d'accélérer le processus. C'est dans ce cadre que s'inscrit ce mémoire. Nous décrirons plus amplement Minamo dans la section suivante.

A côté de ces quatre groupes de recherche scientifique, Cenaero possède une équipe qui s'occupe du centre de calcul de haute performance (HPC). Il est le plus important de la région et s'est classé dans le top 500 des meilleurs calculateurs en juin 2008. Ce super-ordinateur compte aujourd'hui plus de 2000 processeurs, suite à plusieurs mises à jour effectuées depuis 2004. Dans quelques mois, cette capacité va à nouveau être augmentée et le nombre de processeur sera plus que doublé. En plus de l'utiliser pour ses propres calculs, Cenaero le met à disposition de ses clients et partenaires. En effet, ceux-ci peuvent louer des "noeuds" de ce super-calculateur. Cela permet de profiter des performances de l'ordinateur et de sa haute technologie pour un faible coût.

Finalement, Cenaero comporte un dernier groupe de travail : le groupe Business et Project Management. Il est actif dans les projets de recherche et développement mais offre également son aide aux clients pour l'aspect administratif des projets de recherche menés en collaboration avec l'entreprise.

2.2 Le logiciel Minamo

Nous venons de voir que Cenaero est une entreprise qui travaille sur des problèmes liés au monde de l'industrie et des simulations numériques de haute fidélité. Les fonctions manipulées sont donc souvent coûteuses (une seule simulation peut prendre jusqu'à plusieurs heures sur plusieurs processeurs). L'algorithme utilisé pour optimiser ces fonctions est un algorithme génétique. Ce type de méthode demande un grand nombre d'évaluations de la fonction objectif afin de converger. Il est donc difficile d'explorer tout l'espace de conception en utilisant l'expression couteûse tout en restant dans des temps d'exécution raisonnables. De plus, l'expression analytique des fonctions n'est pas toujours connue (fonction "black box"), on n'a alors pas toujours accès à leurs dérivées, ce qui peut rendre difficile en pratique l'utilisation des méthodes de type gradient. La solution est d'utiliser un modèle des fonctions en jeu (i.e., la fonction objectif et les contraintes), qui pourra être évalué à faible coût. C'est cette optimisation assitée par un modèle qu'implémente Minamo. Trois éléments influencent notamment les performances d'une telle méthode : l'algorithme d'optimisation utilisé, le modèle employé et la façon de construire et de gérer ce modèle. Cette section est basée sur des slides de formation au logiciel Minamo ainsi que sur l'article [26].

2.2.1 Principe d'optimisation dans Minamo

Le logiciel Minamo permet d'optimiser une fonction via un algorithme génétique appliqué à un modèle de cette fonction. Afin de construire ce modèle, un *plan d'expériences* (ou DoE pour Design of Experiment) est généré. Il s'agit d'un ensemble de points générés suivant un des processus aléatoires que nous décrirons dans la section suivante. On évalue ensuite ces points via la fonction objectif exacte [1], obtenant ainsi une base de données. L'étape suivante consiste à construire un modèle de la fonction objectif en se basant

1. Il peut éventuellement y avoir plusieurs fonctions objectif (dans le cas de problèmes multi-objectifs) et/ou des contraintes.

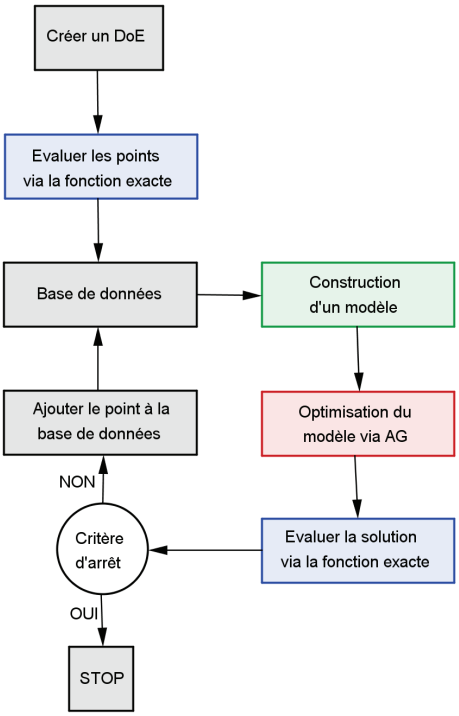


FIGURE 2.1 – Schéma du fonctionnement de Minamo

sur cette base de données. Le but est de construire un modèle qui pourra être évalué à faible coût. On ne sera donc pas limité sur le nombre d'appels que l'on peut y faire. L'algorithme génétique (AG) est ensuite utilisé afin de déterminer l'optimum global de ce modèle. Durant l'optimisation, on utilise uniquement le modèle et pas la fonction exacte, ce qui permet de réduire le temps nécessaire pour obtenir la solution. Dans le but d'améliorer la qualité du modèle au cours de l'algorithme, on évalue le point solution de l'algorithme génétique via la fonction objectif exacte et on ajoute ce nouvel élément à la base de données. On peut éventuellement évaluer d'autres points que l'optimum trouvé, et les ajouter à la base de données. Le modèle construit à l'itération suivante sera alors plus précis puisque construit à partir d'un plus grand nombre de points. Ce processus de modélisation est dit *online* ou adaptatif car le modèle est progressivement amélioré au cours de l'optimisation. Ces différentes étapes sont répétées jusqu'à atteindre le nombre maximum d'itérations qui a été fixé préalablement. Un schéma de ce processus est représenté à la Figure 2.1.

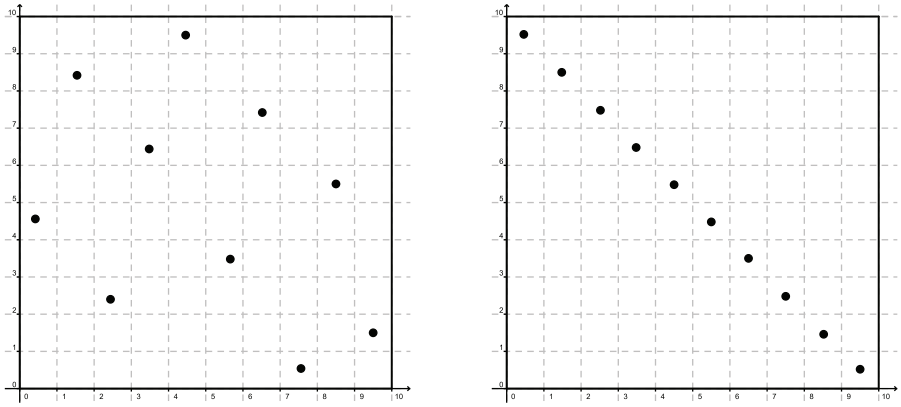
2.2.2 Plan d'expériences

Le plan d'expériences (DoE) constitue la première tâche réalisée par Minamo mais aussi une des plus importantes. Le choix des points qui le constituent va influencer la qualité du modèle qui sera construit par la suite. Le but est de trouver un ensemble de points qui maximise le rapport entre la

précision du modèle obtenu et le nombre de points évalués.

Les techniques classiques de DoE sont appropriées pour des données provenant d'expériences physiques, où les erreurs sont aléatoires. Le *"full factorial"*, le *"central composite"* et le *"Box-Behnken"* sont trois de ces techniques. Comme les fonctions traitées par Minamo proviennent majoritairement de simulations sur ordinateurs (méthodes déterministes), ces techniques ne sont pas les plus adéquates. Minamo utilise donc des techniques dites de remplissage de l'espace. Trois sont implémentées dans le logiciel : Latin Hypercube Sampling (LHS), Centroidal Voronoi Tessellation (CVT) et Latinized CVT (LCVT).

La première méthode, le LHS, est la plus simple. En 2 dimensions, un carré latin est une grille carrée contenant des points tels qu'il y a exactement un point par ligne et par colonne. Pour obtenir un hypercube latin, il suffit de généraliser la propriété à un plus grand nombre de dimensions. L'inconvénient de cette méthode est que les points peuvent ne pas bien recouvrir tout l'espace de conception. L'exemple typique de ce problème est d'obtenir tous les points de l'échantillon situés sur la diagonale du carré. Deux ensembles de points vérifiant la propriété du carré latin sont représentés à la Figure 2.2.



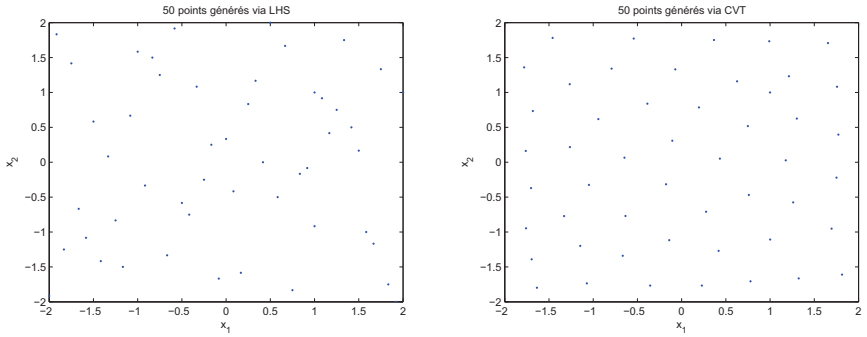
(a) Echantillon LHS

(b) Echantillon LHS avec points sur la diagonale

FIGURE 2.2 – Deux configurations possibles avec LHS pour générer 10 points dans un espace à 2 dimensions.

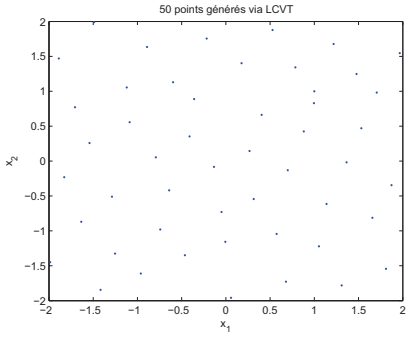
La seconde technique, CVT, se base sur une partition de l'espace en régions. Une région de l'espace est constituée de l'ensemble des points qui sont plus proches d'un générateur que de tous les autres générateurs, un générateur étant un point particulier. L'avantage des CVT par rapport au LHS est que les points ainsi générés couvrent mieux tout l'espace. Par contre, aucun point ne sera généré près du bord de l'espace de recherche par cette méthode. Un autre point faible de cette méthode apparaît lorsqu'on projette les points sur un des axes (ou un plan lorsqu'on travaille à 3 dimensions au moins). Ceux-ci sont regroupés par zones et ne couvrent pas correctement tout l'espace disponible.

La solution pour éviter ces désavantages est de combiner les 2 méthodes précédentes. On obtient ainsi le LCVT. Cela consiste à construire un échan-



(a) Echantillon LHS

(b) Echantillon CVT



(c) Echantillon LCVT

FIGURE 2.3 – Ensembles de 50 points obtenus via les 3 méthodes d'échantillonage.

tillonage de points via la méthode CVT. Cet ensemble de points est ensuite latinisé, i.e., l'ensemble de points est transformé en un ensemble de points voisins qui vérifient la propriété de l'hypercube latin. Cette méthode "mixte" permet de réduire le problème de regroupement (lorsqu'on projette les points sur un axe) de la méthode CVT tout en produisant un échantillon qui recouvre mieux tout l'espace que celui produit par le LHS. C'est donc cette technique qui sera le plus souvent utilisée dans les applications. La Figure 2.3 illustre 50 points obtenus à partir de chacune des 3 méthodes, dans un espace à 2 dimensions et où les variables sont bornées entre -2 et 2.

Les 3 techniques de construction de DoE que nous venons de décrire sont dites "a priori", i.e., l'échantillon de points est construit indépendamment de la fonction à considérer. Une autre façon de procéder est d'employer une technique qualifiée de "a posteriori". Celle-ci tient compte du problème étudié et utilise l'information sur la fonction objectif qui est disponible. Elle permet donc d'obtenir un modèle plus précis de la fonction étudiée. Cette technique se nomme *auto-adaptative DoE*. La première phase revient à générer un ensemble initial de points via une des 3 techniques "a priori" décrites ci-dessus. On construit alors un modèle dont on évalue la validité (cfr. Section 2.2.3). Un ou plusieurs nouveau(x) point(s) est (sont) alors sélectionné(s) et ajouté(s) au DoE. Cette méthode adaptative permet d'augmenter locale-

ment l'intensité de l'échantillon afin de reproduire au mieux la physique du problème considéré. Par exemple, pour une fonction qui présente une zone fortement non linéaire et une autre plus "lisse", on choisira davantage de points dans la zone fortement non linéaire pour que le modèle puisse reproduire au mieux le comportement de la fonction. Cette technique permet également de réduire la probabilité de la chute dans un optimum local de la fonction suite à une mauvaise précision du modèle.

2.2.3 Modèles approchés

L'utilisation de modèles dans l'optimisation a pour but d'accélérer le processus de convergence vers la solution en limitant le nombre d'appels à la fonction exacte (qui peut être coûteuse à évaluer). Cependant, pour obtenir les meilleurs résultats possibles, il est nécessaire que le modèle considéré soit une bonne représentation de la fonction étudiée et suffisamment précis. La difficulté réside dans le fait d'obtenir cette précision tout en n'utilisant qu'un nombre limité d'évaluations de la fonction exacte. Pour des problèmes de grande dimension et fortement multimodaux, les modèles polynomiaux ne sont pas conseillés. C'est pourquoi Minamo utilise d'autres types de modèles [22] : les réseaux de fonctions à base radiale (RBF), les modèles de Kriging et les machines à vecteurs de support (SVM) [16]. Ces modèles ont l'avantage de pouvoir décrire des fonctions multimodales complexes.

Les modèles RBF sont intéressants à utiliser car ils sont rapidement construits et ils donnent de bons résultats pour l'interpolation de fonctions de grande dimension, fortement non linéaire et/ou qui possèdent des données éparpillées. Un modèle RBF fait intervenir des fonctions à base radiale. Il s'agit de fonctions du type

$$h(\|x - c\|_2, \sigma),$$

où c représente un point d'interpolation et σ un paramètre à déterminer. Cette fonction dépend donc de la distance entre le point d'interpolation et celui auquel on veut évaluer la fonction. Suivant la forme prise par la fonction h, on obtient différents types de modèles RBF. Dans Minamo, deux types de modèles sont implémentés : les gaussiens et les multiquadriques. Pour le RBF gaussien, la fonction à base radiale est donnée par

$$h(r, \sigma) = \exp\left(-\frac{r^2}{2\sigma^2}\right),$$

et par

$$h(r, \sigma) = \sqrt{\frac{r^2}{\sigma^2} + 1}$$

pour le RBF Multiquadrique, où $r = \|x - c\|_2$. Lorsque l'ensemble d'interpolation contient plusieurs points c_i, on construit une fonction à base radiale pour chacun. Le modèle RBF est obtenu en prenant une combinaison linéaire des fonctions à base radiale. Il peut s'écrire sous la forme

$$y(x) = \sum_{i=1}^{k} \omega_i \; h_i(\|x - c_i\|_2, \sigma_i),$$

où k est le nombre de points dans l'ensemble d'interpolation, les ω_i sont les poids qui sont déterminés par la résolution de la condition d'interpolation et h_i est la fonction à base radiale qui dépend du point d'interpolation c_i et du paramètre σ_i. Le choix de la fonction à base radiale et des paramètres se fait automatiquement dans Minamo, indépendamment de l'utilisateur. Cela permet de sélectionner le modèle qui correspond le mieux à la fonction objectif.

Un modèle de Kriging peut s'écrire comme la somme de deux termes,

$$y(x) = \mu(x) + Z(x),$$

où $\mu(x)$ représente une tendance globale qui est estimée à partir des données et $Z(x)$ des déviations locales. Il s'agit d'un processus stochastique dans lequel intervient une fonction de corrélation, dépendante de 2 paramètres. Ceux-ci peuvent être déterminés en résolvant un problème d'estimation du maximum de vraissemblance. Cela revient à résoudre un problème d'optimisation où la fonction objectif est multidimensionnelle et multimodale. L'algorithme génétique implémenté dans Minamo est utilisé pour déterminer la solution de ce problème. Deux types de modèles Kriging sont disponibles dans Minamo : le Kriging ordinaire, obtenu lorsque $\mu(x)$ est égal à une constante, et le Kriging universel lorsque la fonction $\mu(x)$ est obtenue via une régression linéaire. La construction des modèles Kriging fait intervenir une étape d'optimisation, et peut dès lors devenir fort coûteuse. Mais ces modèles permettent d'obtenir une prédiction globale des valeurs de la fonction objectif et une estimation de l'incertitude de la prédiction aux points inobservés.

La qualité du modèle considéré est très importante afin de maximiser les chances de converger vers un optimum de la fonction objectif exacte. C'est pourquoi une procédure qui permet d'évaluer la précision d'un modèle a été implémentée dans Minamo. Il s'agit de la procédure de validation "Leave-One-Out" (LOO) qui permet d'estimer la qualité du modèle sans devoir créer de nouvel ensemble de points pour effectuer la validation. Supposons qu'on dispose d'un ensemble de k points. On sélectionne $k-1$ points à partir desquels on construit un modèle. Celui-ci est ensuite utilisé pour évaluer le point qui n'a pas été utilisé pour sa construction. On répète l'opération k fois, en laissant de côté un point différent à chaque fois. On peut alors comparer les valeurs estimées (via l'évaluation du modèle) aux valeurs réelles en calculant un coefficient de corrélation. Si ce dernier est proche de 1, cela signifie que le modèle représente la fonction exacte avec précision.

Nous avons vu que la gestion des modèles était un point important en ce qui concerne les performances de l'optimisation. C'est pourquoi une stratégie de *Move-Limit* a été implémentée dans Minamo. Cette procédure permet de faire varier les bornes sur les variables au cours de l'optimisation, ce qui revient à restreindre l'espace de recherche à une certaine région. On peut ainsi garantir qu'on ne génère pas de point dans une région où le modèle n'est pas valable en se limitant dans une région "de confiance" où le modèle reste valide. On exploite des modèles locaux. L'idée est d'agrandir la région de recherche lorsqu'une amélioration parmi la population est constatée, car cela signifie que le modèle est bon. Au contraire, on réduira la région de recherche si aucune amélioration n'est constatée. On peut choisir d'appliquer

cette procédure dès le début de l'optimisation ou seulement après un certain nombre d'itérations. Le choix de ce nombre d'itérations déterminera l'importance accordée à l'exploration de l'espace (itérations sans Move-Limit, i.e., on explore tout l'espace) et à l'exploitation de l'information présente plus localement (itérations avec Move-Limit, on se restreint à une région autour de l'optimum courant).

2.2.4 Algorithme génétique de Minamo

La méthode d'optimisation implémentée dans Minamo est un algorithme génétique. Ce choix est justifié par les différentes capacités d'un algorithme génétique. Rappelons-en quelques-unes présentées dans le chapitre précédent : c'est une méthode d'ordre 0, qui est efficace dans la détermination de l'optimum global du problème ; elle permet l'optimisation de fonction mono- et multi-objectifs ; elle est robuste et peut traiter des fonctions bruitées, discontinues, fortement non linéaires, etc. Le point faible de ces algorithmes étant le grand nombre d'évaluations de la fonction objectif nécessaire pour obtenir la convergence de la méthode, l'utilisation des méta-modèles est cruciale.

L'algorithme implémenté dans Minamo suit le schéma général présenté au Chapitre 1. Rappelons qu'il utilise un codage réel des variables. Au niveau de la sélection, c'est la méthode de tournoi qui a été choisie pour éviter les problèmes de variance et de scaling. L'élitisme est également appliqué à quelques individus. Au niveau de l'étape de variation, ce sont des opérateurs de cross-over et de mutation qui sont appliqués aux individus. Pour définir la fonction de fitness, on utilise la fonction objectif lorsqu'il n'y a pas de contraintes.

2.2.5 Autres particularités de Minamo

Pour terminer ce chapitre, nous présentons quelques autres caractéristiques de Minamo : l'utilisation d'une fonction de mérite, un aperçu de la gestion des contraintes (qu'elles soient d'inégalité et/ou d'égalité), la façon de gérer les échecs de simulation et l'optimisation multi-objectif.

Une *fonction de mérite* de Torczon et Trosset [31] est implémentée dans Minamo. Elle est de la forme

$$merit(x) = y(x) - \rho d(x),$$

où $y(x)$ correspond au modèle, $\rho \geq 0$ et $d(x) = \min_i \|x - c_i\|_2$ avec les c_i qui sont les points ayant servi à la construction du modèle $y(x)$. Le but est de maintenir une certaine diversité de la population. En considérant la distance entre un individu et les autres individus et en favorisant les candidats isolés des autres, on peut effectuer une recherche aux endroits indiqués comme minima potentiels par le modèle, mais aussi explorer la fonction dans les zones peu connues.

Une première façon de traiter les contraintes est d'utiliser une *fonction de pénalité*. C'est la façon la plus courante car la plus simple à implémenter. La fonction de pénalité est nulle aux points admissibles et strictement positive

pour des points qui violent les contraintes. Elle est ajoutée à la fonction objectif moyennant des paramètres de pénalité. Les solutions non admissibles sont ainsi pénalisées. Le désavantage de cette méthode est la difficulté d'estimation des paramètres de pénalité. C'est pourquoi une méthode qui ne fait pas intervenir de paramètre a été proposée pour gérer les contraintes dans l'algorithme génétique : la méthode *Constraint Tournament Selection*. Elle consiste à comparer 2 individus : celui qui est admissible est toujours préféré à un individu non admissible ; de deux individus admissibles, on choisira celui qui possède la meilleure fitness ; si aucun individu n'est admissible, on garde celui qui viole le moins les contraintes.

La gestion des échecs de simulation est importante pour assurer un bon comportement de l'optimisation. Au lieu de filtrer les résultats qui n'ont pas convergé, Minamo va utiliser cette information pour rediriger la recherche vers des zones plus "stables". Le succès ou l'échec de la simulation est enregistré pour chaque point. On ne construit plus un mais deux modèles : un modèle qui prédit la réponse de la fonction objectif et un autre qui prédit le succès ou l'échec d'une simulation, la région instable. Il est donc important que les indicateurs de succès/échec soient exacts puisqu'ils sont exploités par l'algorithme.

L'optimisation multi-objectif correspond à optimiser plusieurs fonctions objectif en même temps. La difficulté réside dans le fait que les différents objectifs sont souvent contradictoires. Il existe deux types de méthodes pour résoudre ce problème. Le premier, les méthodes *a priori*, consiste à transformer le problème multi-objectif en mono-objectif. On combine les différentes fonctions à optimiser en une seule via une somme pondérée. Le choix des poids qui doit se faire avant d'optimiser justifie le nom de la méthode. C'est une technique facile à implémenter puisqu'on peut utiliser les méthodes d'optimisation mono-objectif. Par contre, le choix des poids peut être difficile, particulièrement quand les différents objectifs sont fortement contradictoires. Une autre façon de procéder est d'appliquer une méthode qualifiée d'*a posteriori*. Les différents objectifs sont gardés séparés durant l'optimisation. On ne trouve pas une unique solution mais une famille de candidats qui représentent des compromis entre les différents objectifs, c'est le front de Pareto. On choisit alors une solution parmi cet ensemble, après la phase d'optimisation. Minamo permet d'utiliser les deux méthodes.

Chapitre 3

Motivations du couplage entre méthode globale et locale

La question du couplage entre méthode globale et locale est notamment abordée dans la littérature par Tenne et Armfield dans trois articles (cfr. [28], [29] et [30]). Les deux premières sections de ce chapitre sont consacrées à la présentation de leur point de vue, sur base des trois articles cités ci-dessus. Dans un premier temps, nous présentons les motivations des auteurs pour construire une telle méthode. Nous décrivons ensuite deux algorithmes qu'ils présentent dans ces trois articles. Une autre approche du couplage d'un algorithme génétique et d'une méthode locale est présentée dans l'article [21]. Un bref résumé en est présenté dans la troisième section de ce chapitre. Finalement, nous clôturons ce chapitre par la présentation de la forme de couplage que nous avons choisi d'appliquer afin de construire nos propres méthodes hybrides.

3.1 Pourquoi coupler ?

Les articles de Tenne et Armfield étudient l'optimisation dans un cadre bien précis. Ils se focalisent sur l'optimisation de fonctions coûteuses, résultant de simulations haute fidélité effectuées sur ordinateur. Il s'agit du type de problème qui nous intéresse. Plusieurs problèmes apparaissent lorsqu'on traite de telles fonctions : l'expression analytique de la fonction n'est pas nécessairement connue ; le nombre d'évaluations de la fonction objectif autorisé est limité (à cause du coût d'une évaluation) ; la fonction peut être multimodale et bruitée ainsi qu'indéfinie ou discontinue en certains points (où la simulation numérique n'a pas convergé).

Les algorithmes génétiques (cfr. Section 1.3) possèdent plusieurs avantages. Ils n'utilisent pas de dérivées, sont robustes, permettent une *exploration* globale du domaine admissible (les minima locaux sont évités) et ils obtiennent de bons résultats dans la résolution de problèmes difficiles (fortement multimodaux par exemple). Ils peuvent donc être employés pour optimiser une fonction du type décrit précédemment. Cependant, ils présentent aussi un point faible : beaucoup d'évaluations de la fonction objectif sont nécessaires pour réaliser l'optimisation. Les auteurs citent 3 moyens de remédier

à cette faiblesse. Le premier est *l'héritage de la fitness*[1] i.e., seule la fitness de certains nouveaux individus est calulée tandis que les autres nouveaux individus conservent la fitness de leur parent. La seconde technique consiste à utiliser une *fidélité variable* dans la précision de la simulation. Enfin la dernière méthode revient à effectuer une *optimisation assistée par un modèle*. C'est cette dernière piste qui a été retenue dans les trois articles [28, 29, 30] afin de définir le nouvel algorithme. Il s'agit également de la stratégie qui est adoptée dans Minamo. Le modèle est une approximation mathématique de la fonction objectif exacte, qui peut être évalué à faible coût. Le modèle intervient aussi bien au niveau de l'algorithme génétique qu'au cours de la méthode locale. Cependant, différents modèles peuvent être employés dans les deux phases, pour représenter la fonction plutôt globalement ou de façon plus locale par exemple.

De par leur capacité d'exploration, les algorithmes génétiques ont de grandes chances de déterminer la région qui contient l'optimum global du problème, en évitant au maximum les solutions locales. L'exploration est due aux différents choix aléatoires qui sont réalisés dans l'algorithme. Mais en fin d'optimisation, l'aléatoire n'apporte plus que peu de gain. Il est alors plus intéressant d'utiliser l'information sur le comportement local de la fonction afin d'atteindre l'optimum avec le plus de précision possible. L'application d'une méthode de recherche locale permet ainsi l'*exploitation* de cette information locale, ce qui implique une accélération de la convergence vers la solution optimale.

La combinaison d'un algorithme génétique et d'une méthode de recherche locale résulte en un *algorithme mémétique* ou *algorithme hybride*. La recherche locale peut aussi bien être utilisée en tant qu'opérateur de l'algorithme génétique que comme une étape à part entière. Une telle combinaison de méthodes permet donc d'accélérer la convergence de l'algorithme et de donner une approximation précise de l'optimum en peu d'évaluations de la fonction coûteuse (grâce à l'utilisation du modèle). La répartition du nombre d'évaluations de la fonction coûteuse entre l'algorithme génétique (pour l'exploration) et la méthode locale (pour l'exploitation) est un élément clé de l'efficacité de tout algorithme hybride.

3.2 Quels types d'algorithmes ?

Les algorithmes mémétiques assistés par des modèles sont nombreux. Ils dépendent du choix du modèle, de la méthode locale appliquée, de la façon de combiner cette dernière avec l'algorithme génétique, etc. Les types de modèles les plus souvent utilisés sont les moindres carrés quadratiques, les réseaux de neurones, les modèles de Kriging et les réseaux de fonctions à base radiale (RBF). Il est important que le modèle soit suffisamment précis pour assurer la convergence de l'algorithme vers un vrai optimum, i.e., un optimum de la fonction objectif exacte et non du modèle. La génération d'un

1. Dans le cas d'un problème de maximisation sans contrainte, la fitness correspond à la fonction objectif. Lorsqu'il s'agit d'un problème de minimisation, la fitness représente l'inverse de la fonction objectif.

modèle précis n'est pas toujours évidente, particulièrement quand le nombre de points disponibles pour sa construction est petit.

Deux algorithmes hybrides différents sont proposés dans les articles [28, 29, 30]. Le couplage des méthodes se fait "en dehors" de l'algorithme génétique, i.e., ce dernier est indépendant de la recherche locale. Celle-ci a lieu une fois l'algorithme génétique terminé. Un autre point commun aux deux algorithmes est le choix du type de métamodèle : un RBF. Ce type de modèle est plus précis qu'un simple modèle quadratique pour des fonctions "compliquées" (fortement multimodales par exemple). Par contre, les deux stratégies hybrides diffèrent dans le choix de la méthode locale, mais surtout dans la manière de la combiner avec l'algorithme génétique.

3.2.1 Premier algorithme hybride

La particularité de cet algorithme hybride est qu'il utilise deux modèles différents : un modèle global et un modèle local. Une première optimisation est effectuée sur le modèle global via un algorithme mémétique. Ce dernier intervient ensuite dans l'optimisation du modèle local, au niveau de la résolution du sous-problème de la région de confiance. Le schéma général de cet algorithme hybride est représenté à la Figure 3.1. Il s'agit donc d'un algorithme hybride qui contient une autre méthode hybride. Pour plus de clarté dans la suite de cette section, nous qualifierons d'interne la méthode mémétique qui est appliquée deux fois tandis que l'algorithme hybride représenté à la Figure 3.1 sera désigné par l'adjectif externe.

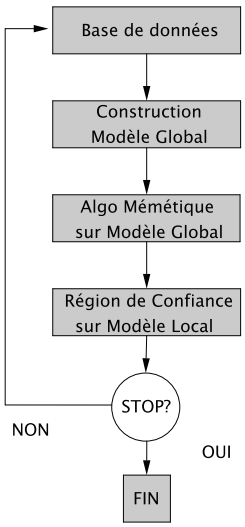


FIGURE 3.1 – Schéma général du premier algorithme hybride (externe).

La première étape de l'optimisation consiste en la construction du modèle global. On construit donc un modèle RBF à partir d'un petit nombre de points. Le but est d'obtenir un "aspect du paysage" de la fonction sans dépenser trop d'appels à la fonction coûteuse. Différents choix, comme celui du

type de la fonction à base radiale (gaussien, multiquadrique, multiquadrique inverse ou linéaire) et des hyper-paramètres, doivent être effectués lors de la constuction. Ces différents choix sont réalisés par l'utilisateur. Pour plus de détails sur la construction du modèle, on pourra se référrer à [29].

L'algorithme mémétique interne est ensuite lancé sur le modèle global. Cet algorithme hybride interne combine un algorithme génétique à codage réel, qui se termine si 10 générations successives ne subissent plus aucune modification, avec une recherche locale SQP (Sequential Quadratic Programming). La recherche locale prend comme point initial la solution fournie par l'algorithme génétique et utilise l'algorithme quasi-Newton BFGS. Cette méthode ainsi que le SQP sont présentés dans [11].

Finalement, la phase d'optimisation locale est initiée à partir de la solution optimale du modèle global. Cette dernière étape utilise une méthode de région de confiance (cfr. Section 4.1) dont le modèle est un RBF construit localement, i.e., calculé uniquement à partir de points se trouvant à l'intérieur de la région de confiance. Le mécanisme de la région de confiance, associé au modèle local, permet de garantir l'efficacité de la méthode ainsi que sa convergence vers un optimum de la fonction objectif exacte. L'optimisation du modèle (local) à l'intérieur de la région de confiance est réalisée en appliquant l'algorithme hybride interne décrit juste avant. Les étapes d'acceptation du nouvel itéré et de mise à jour du rayon de la région de confiance sont légèrement différentes de celles du schéma classique de région de confiance car, dans le cas présent, le nombre de points de la région de confiance qui servent à la construction du modèle joue un rôle. La recherche locale possède deux critères d'arrêt : une borne inférieure sur le rayon de la région de confiance et un nombre maximal d'évaluations de la fonction objectif.

Ces trois étapes forment une itération de l'algorithme hybride externe. Elles sont répétées jusqu'à ce que le nombre maximal d'évaluations de la fonction objectif soit atteint. En plus de cette description de l'algorithme, les articles proposent également une méthode qui permet d'améliorer la précision d'un modèle. Cela se révèle très utile lors de la construction du modèle local qui peut être de moins bonne qualité car uniquement construit sur des points appartenant à la région de confiance. Des résultats de tests numériques effectués sur cette méthode hybride peuvent être consultés dans les articles [28] et [29].

3.2.2 Second algorithme hybride

Le second algorithme hybride proposé par Tenne et Armfield dans [30] combine un algorithme génétique et une méthode locale de type région de confiance. Sa particularité réside dans la façon de déterminer quand passer d'une méthode à l'autre. L'outil principal utilisé pour fixer ce choix est basé sur le critère de Métropolis que nous décrivons dans la suite de cette section.

La première étape de l'optimisation consiste toujours à construire un modèle de type RBF. La différence par rapport au premier algorithme hybride est que l'utilisateur ne doit plus sélectionner le type de fonction ni les hyper-paramètres. Une procédure de sélection automatique (basée sur un

critère de type LOO) est implémentée, ce qui permet de choisir le modèle qui correspond le mieux à la fonction objectif [2]. Le modèle RBF ainsi obtenu est combiné à une fonction de pénalité [3] afin d'obtenir le modèle qui sera considéré durant l'optimisation. La fonction de pénalité consiste à pénaliser les points où la simulation exacte (fonction objectif) n'a pas convergé. Cela permet d'éviter ce type de points au cours de l'optimisation.

L'optimisation du modèle RBF au moyen de l'algorithme génétique constitue la deuxième étape. Il s'agit d'un algorithme génétique classique, à codage réel. On détermine ainsi x_{AG}, l'optimum global du modèle construit à l'étape précédente. L'algorithme est stoppé quand le nombre maximum de générations est atteint ou si x_{AG} est meilleur que tous les candidats évalués jusqu'alors et est entouré par suffisamment de points (cela indique la convergence).

L'étape suivante permet de décider si une recherche locale est appliquée à partir de x_{AG}. En effet, il n'est pas toujours intéressant d'initier une telle recherche sur la solution de l'algorithme génétique. Si l'image par la fonction objectif de x_{AG} est strictement inférieure à toutes les valeurs trouvées jusqu'alors, cela signifie que le modèle est suffisamment précis et que cet individu est un bon candidat pour initier une recherche locale. Cette recherche peut alors utiliser tous les appels restant à la fonction objectif. Au contraire, si l'image de x_{AG} n'est pas la meilleure, on peut supposer que le modèle n'est pas suffisamment précis. Dans ce cas, on n'effectue pas de recherche locale ou, si on en autorise une, on ne lui permet pas d'utiliser tous les appels à la fonction objectif encore disponibles.

C'est une règle basée sur le critère de Métropolis qui va permettre de déterminer le nombre d'évaluations de la fonction objectif attribué à la méthode locale. Le critère de Métropolis permet de déterminer une probabilité en fonction de deux paramètres : une variation d'énergie et une température. Son expression est donnée par

$$p(\Delta E, T) = \begin{cases} 1 & \text{si } \Delta E < 0, \\ \exp\left(\dfrac{-\Delta E}{T}\right) & \text{si } \Delta E \geq 0. \end{cases}$$

On obtient une valeur dans l'intervalle $[0, 1]$. Ce critère peut donc servir à déterminer une proportion. L'adaptation du critère au calcul du nombre d'évaluations de fonction attribué à la méthode locale est simple. Notons ce nombre m_{loc} et définissons-le comme une proportion du nombre d'évaluations encore disponibles $(m_{tot} - m_{cur})$ (avec m_{tot} le nombre maximal autorisé et m_{cur} le nombre déjà utilisé) :

$$m_{loc} = l(m_{tot} - m_{cur}) \quad 0 \leq l \leq 1.$$

La valeur de l s'obtient en appliquant le critère de Métropolis où ΔE correspond à la différence entre l'image de x_{AG} et la meilleure valeur trouvée jusqu'alors, et T représente le rapport entre m_{cur} et m_{tot}. Si la solution de l'algorithme génétique est meilleure que ce qui avait été trouvé jusqu'alors,

2. C'est aussi le cas dans Minamo.
3. Un mécanisme similaire est également implémenté dans Minamo.

"ΔE" sera négatif, l vaudra 1 et toutes les évaluations de fonction seront laissées à la méthode locale. Dans le cas contraire, seulement une partie des évaluations sera utilisée par la recherche locale et une optimisation globale sera à nouveau effectuée ensuite.

La dernière étape correspond à l'application de la recherche locale. Celle-ci peut ne pas avoir lieu, suite au résultat du critère de Métropolis. Il s'agit d'une méthode de région de confiance classique, dans le sens où le modèle utilisé est une approximation quadratique de la fonction objectif. Le modèle est construit via une interpolation multivariée lagrangienne (algorithme de Sauer-Xu [27]) et l'optimum à l'intérieur de la région de confiance est déterminé grâce à une méthode SQP. Les avantages de cette méthode locale sont qu'elle ne calcule pas de dérivée par différences finies et le problème quadratique est résolu efficacement. Pour renforcer le caractère local de cette étape, seuls les points appartenant à un voisinage de la région de confiance sont pris en compte lors de la construction du modèle. La méthode locale possède plusieurs critères d'arrêt. L'algorithme se termine si le rayon de la région de confiance est inférieur à une valeur minimale fixée, ou si la norme du gradient projeté (du modèle) sur l'ensemble admissible est proche de zéro, ou si le nombre d'évaluations de la fonction objectif autorisé pour la recherche locale est atteint, ou si on ne peut pas trouver de point défini [4] à l'intérieur de la région de confiance.

Si, à la fin de la phase locale, le nombre maximal d'évaluations de la fonction objectif n'est pas atteint, on ajoute les nouveaux points calculés à la base de données et on recommence une nouvelle itération de l'algorithme hybride. Celui-ci s'arrête lorsqu'une valeur "suffisamment bonne" de la fonction objectif est trouvée (par rapport à une borne fixée dès le début) ou quand toutes les évaluations de fonction disponibles ont été utilisées. Les principales étapes de cet algorithme mémétique sont schématisées à la Figure 3.2.

3.3 Quelles autres formes de couplage ?

Les algorithmes mémétiques présentés précédemment combinent les différentes méthodes en les appliquant l'une après l'autre. Un autre point de vue sur les méthodes hybrides est donné dans l'article *Hybrid Genetic Algorithms : A Review* [21]. Cet article étudie différents moyens d'améliorer un algorithme génétique en lui "greffant" d'autres méthodes d'optimisation.

La clé de l'efficacité d'un algorithme génétique est un bon équilibre entre l'exploration et l'exploitation. Utiliser une recherche locale au sein même de l'algorithme génétique peut permettre d'améliorer l'exploitation sans limiter ses capacités d'exploration. L'idée est la même que celle présentée par Tenne et Armfield. L'algorithme génétique est utilisé pour la recherche initiale car il donne une vue globale de l'espace admissible. La méthode locale est appliquée sur la population finale d'une *génération* de l'algorithme génétique. Elle amène de la diversité dans les gènes, ce qui permet d'éviter la convergence

4. Un point défini est un point où la simulation exacte qui calcule la valeur de la fonction objectif a convergé et donné un résultat exploitable.

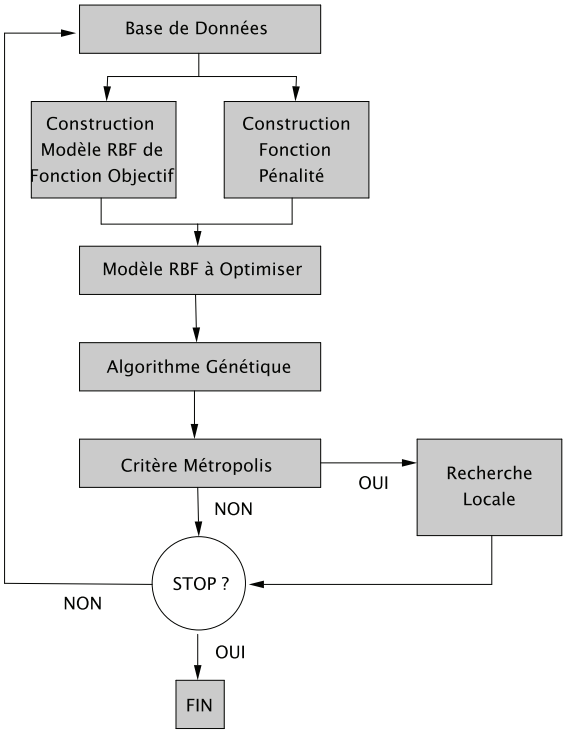


FIGURE 3.2 – Schéma général du second algorithme hybride.

prématurée et d'améliorer la qualité des solutions. Dans le cas des problèmes fortement contraints, les opérateurs génétiques de mutation et de croisement peuvent conduire à des solutions non-admissibles. Cela implique une perte de temps et un risque de convergence rapide quand trop de candidats sont non-admissibles. La recherche locale peut être utilisée pour "réparer" les solutions non-admissibles produites en déterminant des points admissibles.

La façon d'utiliser l'information locale dans un algorithme hybride et l'équilibre entre la recherche globale et la recherche locale (i.e., exploration - exploitation) sont deux facteurs qui influencent les performances de la méthode hybride. La recherche locale peut modifier un individu au niveau de sa structure génétique et de sa fitness. Mais il est également possible de ne modifier que la fitness d'un individu en laissant son patrimoine génétique intact. Dans ce cas, l'information gagnée ne sera pas transmise aux générations futures. Une combinaison des deux points de vue peut aussi être envisagée. En ce qui concerne l'équilibre entre exploration et exploitation, trois paramètres interviennent : la fréquence de la recherche locale, sa durée et sa probabilité. Un algorithme hybride classique applique une méthode locale après chaque génération (la fréquence vaut alors 1) mais une autre fréquence peut être envisagée, voire même une fréquence variable au cours de l'algorithme. La durée de la recherche locale dépend du problème considéré. Mais une recherche partielle, i.e., qui ne dure pas jusqu'à obtenir une convergence, est moins risquée, que ce soit en terme de temps, de coût, ou de perte de diver-

sité. Finalement, la méthode locale peut être appliquée à toute la population ou seulement à une portion de celle-ci. Différentes méthodes existent pour déterminer cette fraction de la population, pour plus de détails sur celles-ci on se réferrera à l'article [21].

En combinant ainsi l'algorithme génétique avec une méthode locale, les performances de l'algorithme génétique peuvent être améliorées à plusieurs niveaux. La méthode hybride a une capacité de recherche qui est supérieure à celle d'un algorithme génétique pur. La qualité des solutions est améliorée grâce aux capacités de la méthode locale. Le fait d'appliquer une méthode locale sur la population finale d'une génération permet de représenter équitablement chaque zone de l'espace de recherche (en en déterminant les optima globaux), ce qui évite une convergence prématurée qui détériore la qualité des solutions. L'application d'une méthode locale permet aussi d'accélérer la convergence vers une solution car les algorithmes génétiques sont rapides pour trouver la zone qui contient l'optimum global d'un problème mais sont lents pour atteindre cette solution avec précision. L'efficacité de l'algorithme hybride d'un point de vue du temps d'exécution de l'algorithme peut ainsi être améliorée. Il est également signalé dans l'article qu'il existe des moyens de travailler avec une population de taille inférieure lorsqu'une recherche locale est utilisée. On obtient ainsi un gain en mémoire.

3.4 Quels choix pour ce mémoire ?

Nous avons choisi de construire un algorithme mémétique dont la structure est assez similaire à celle de la méthode hybride présentée à la Section 3.2.2. Dans un premier temps, un algorithme génétique est utilisé. Une méthode locale est ensuite appliquée. Cependant, nous avons choisi de n'effectuer qu'une seule itération externe. Cela signifie qu'on utilise une seule fois chaque méthode, sans repasser à l'algorithme génétique après avoir terminé la méthode locale. Ce choix permet de tester dans un premier temps le gain que peut apporter la combinaison des deux méthodes. L'étape suivante dans l'étude du couplage entre les deux méthodes serait de passer à plusieurs itérations de l'algorithme hybride. Cela pourrait être l'objet d'un travail ultérieur dans la continuité de ce mémoire. Le schéma général de notre méthode hybride est représenté à la Figure 3.3.

La génération du DoE se fait au moyen des processus implémentés dans Minamo. Pour avoir la meilleure représentation possible de l'espace de recherche, l'échantillon de points est obtenu grâce à la méthode LCVT (cfr. Section 2.2.2).

Dans la phase globale, le modèle optimisé par l'algorithme génétique de Minamo est de type RBF (cfr. Section 2.2.3). Comme dans l'exemple de Tenne et Armfield que nous avons présenté à la Section 3.2.2, le type de la fonction à base radiale et les hyper-paramètres sont choisis automatiquement sur base d'une procédure de LOO. Au cours des itérations, les points évalués via la fonction objectif exacte sont ajoutés à la base de données. On peut ainsi construire un nouveau modèle RBF qui représentera mieux la fonction objectif. Le critère d'arrêt que nous utilisons est un nombre maximal d'évaluations de la fonction objectif. Ce nombre maximal sera discuté

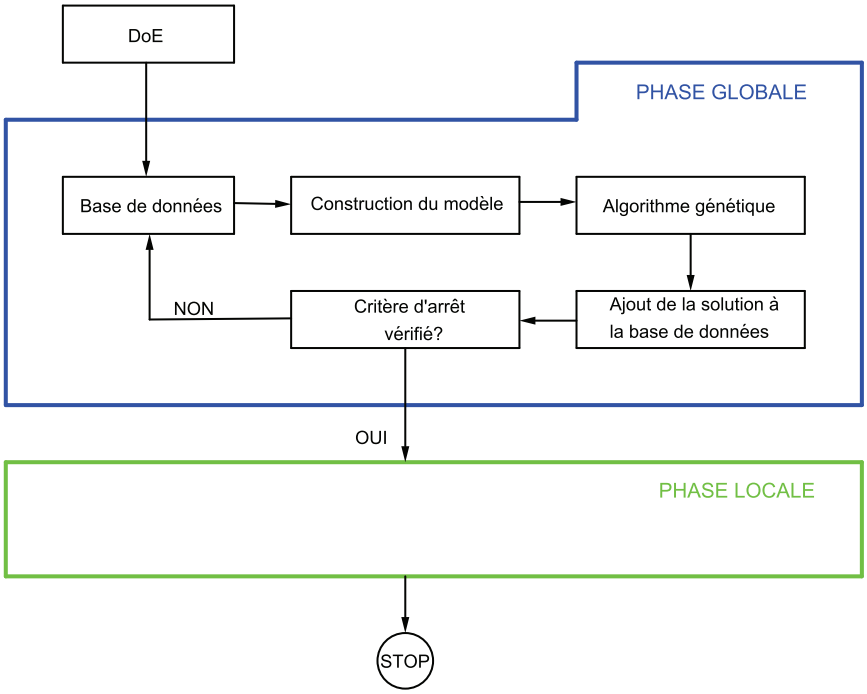


FIGURE 3.3 – Schéma général de notre algorithme hybride.

au Chapitre 5 lors de la discussion sur les valeurs des différents paramètres qui interviennent dans notre méthode hybride.

En ce qui concerne la phase locale, nous ne disposons pas encore des éléments nécessaires pour en réaliser le schéma. De plus, nous allons utiliser plusieurs méthodes locales dont les schémas ne sont pas identiques. Cela nous permettra de construire différents algorithmes hybrides dont nous pourrons ensuite comparer les performances. La description des méthodes locales fait l'objet du chapitre qui suit. A la fin de ce dernier, nous reviendrons sur le schéma de notre méthode hybride et le contenu de la phase locale.

Chapitre 4

Description des méthodes de recherche locale

Dans ce chapitre, nous décrivons les méthodes de recherche locale qui interviennent dans les couplages que nous avons développés. Il s'agit de quatre méthodes de type région de confiance. Les modèles de la fonction objectif qu'elles utilisent sont construits par interpolation. Dans le cas des deux premières méthodes, il s'agit de modèles RBF. Dans la première méthode, le modèle RBF sera à son tour modélisé, cette fois par un polynôme quadratique calculé à partir des dérivées du modèle RBF. Les deux autres méthodes, DFO et BCDFO, utilisent des modèles de type polynomiaux. Ces quatres méthodes n'utilisent jamais les dérivées de la fonction objectif exacte, ce qui les rend applicables à des problèmes dont les dérivées ne sont pas disponibles. Pour chaque méthode, nous présentons son fonctionnement ainsi que les questions liées à la construction du modèle et à sa minimisation. Nous abordons également l'aspect de l'implémentation des algorithmes. Nous terminons ce chapitre par un rappel des éléments importants qui composent nos méthodes hybrides.

4.1 L'algorithme de région de confiance

Comme nous l'avons présenté au Chapitre 1, les méthodes de région de confiance sont un moyen de "globalisation", dans le sens où elles permettent d'obtenir la convergence de l'algorithme à partir de n'importe quel point initial. Mais il s'agit surtout de méthodes locales, grâce à l'utilisation de la région de confiance qui restreint la recherche dans un voisinage du meilleur candidat trouvé jusqu'alors.

Nous commençons par présenter le fonctionnement général d'un algorithme de région de confiance. Celui-ci servira de base à nos différentes implémentations. Nous discutons ensuite le choix du modèle utilisé et présentons les méthodes pour résoudre le sous-problème dans la région de confiance. Enfin, nous décrivons les deux méthodes que nous avons implémentées.

4.1.1 L'algorithme général

L'algorithme que nous présentons ici a pour but la résolution d'un problème du type

$$\min_{x \in \mathbb{R}^n} f(x), \tag{4.1}$$

où x est un vecteur à n composantes réelles et $f : \mathbb{R}^n \to \mathbb{R}$ est une fonction à valeurs réelles, deux fois différentiable [1]. A chaque itération k de l'algorithme, on définit un modèle de la fonction f, noté $m_k(.)$, autour de l'itéré courant x_k. On suppose que ce modèle approxime suffisamment bien la fonction dans un voisinage de l'itéré, que l'on nomme la *région de confiance*. C'est un ensemble de points défini par

$$\mathcal{B}_k = \{x \in \mathbb{R}^n \mid ||x - x_k|| \leq \Delta_k\},$$

où Δ_k est le rayon de la région de confiance et $\|.\|$ la norme euclidienne [2] sur $\mathbb{R}^n$. Une fois le modèle et la région de confiance définis, on cherche un pas s_k qui permettra de produire un nouveau candidat itéré $x_k + s_k$. Le pas est déterminé de façon à obtenir une réduction de la valeur prise par le modèle, tout en restant dans la région de confiance, c'est-à-dire $||s_k|| \leq \Delta_k$. Il faut ensuite choisir d'accepter ou de rejeter ce nouveau candidat. Pour cela, on calcule la valeur de la fonction objectif en ce point, $f(x_k + s_k)$, ainsi que la valeur du modèle, $m_k(x_k + s_k)$. On peut alors regarder si la réduction prédite par le modèle est suffisamment proche de celle réalisée par la fonction. Si c'est le cas, le nouveau candidat est accepté et on peut envisager d'agrandir la région de confiance ou de conserver la même. Dans le cas contraire, le candidat $x_k + s_k$ est rejeté et on restreint la région de confiance en diminuant son rayon. On espère se concentrer sur une région où le modèle représente mieux la fonction objectif et ainsi obtenir de meilleures prédictions. L'Algorithme 4.1 reprend, sous forme schématique, ces différentes étapes.

Algorithme 4.1. (Algorithme général de région de confiance)

1. Initialisation :
   Donner x_0 un point initial et Δ_0 un rayon initial de la région de confiance. Donner les constantes η_1, η_2, γ_1 et γ_2 qui satisfont

   $$0 < \eta_1 \leq \eta_2 < 1 \text{ et } 0 < \gamma_1 \leq \gamma_2 < 1.$$

   Calculer $f(x_0)$ et poser $k = 0$.
2. Définition du modèle :
   Choisir $||.||$ et définir un modèle m_k dans la région de confiance $\mathcal{B}_k$.
3. Calcul du pas :
   Calculer un pas s_k qui "décroît suffisamment le modèle" m_k et tel que $x_k + s_k \in \mathcal{B}_k$.

1. Cependant, dans la suite du travail nous n'utiliserons jamais explicitement ces dérivées. Les méthodes développées pourront ainsi être appliquées à des fonctions dont les valeurs des dérivées ne peuvent être calculées.

2. Il est possible de considérer d'autres normes ainsi qu'une norme qui dépend de l'itération k, cfr. [5].

4. Test du nouveau point : Calculer $f(x_k + s_k)$ et définir

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{4.2}$$

Si $\rho_k > \eta_1$, définir $x_{k+1} = x_k + s_k$, sinon définir $x_{k+1} = x_k$.

5. Mise à jour du rayon :

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty[& \text{si } \rho_k \geq \eta_2, \\ [\gamma_2 \Delta_k, \Delta_k] & \text{si } \rho_k \in [\eta_1, \eta_2[, \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{si } \rho_k < \eta_1. \end{cases}$$

Poser $k = k + 1$ et retourner au pas 2.

Le choix du modèle (étape 2) fait l'objet de la Section 4.1.2. La troisième étape, la résolution du sous-problème, est influencée par le choix du modèle réalisé à l'étape précédente. Cette résolution sera discutée à la Section 4.1.3. La question de la mise à jour du rayon de la région de confiance est abordée à la Section 4.1.4. Enfin, l'Algorithme 4.1 ne mentionne aucun critère d'arrêt. Dans notre cas, ceux-ci varient suivant les différentes implémentations. C'est pourquoi nous discutons le choix de ces critères ultérieurement, lors de la description détaillée de nos implémentations aux Sections 4.1.5 et 4.1.6.

4.1.2 Le choix du modèle

Le modèle le plus couramment utilisé dans l'algorithme de région de confiance est une approximation quadratique de la fonction objectif. Celle-ci est de la forme

$$m_k(x_k + s) = m_k(x_k) + \langle g_k, s \rangle + \frac{1}{2} \langle s, H_k s \rangle, \tag{4.3}$$

où $m_k(x_k) = f(x_k)$, $g_k = \nabla f(x_k)$ et H_k est une approximation symétrique de $\nabla^2 f(x_k)$. L'avantage du modèle quadratique est que son optimum peut être déterminé efficacement via des méthodes spécifiques aux problèmes quadratiques.

Une autre façon de modéliser la fonction objectif est d'utiliser un réseau de fonctions à base radiale (RBF), cfr. Section 2.2.3. Ce type de modèle est plus général que l'approximation quadratique. Il permet de représenter plus globalement la fonction tandis qu'un modèle quadratique reste une approximation plus locale de la fonction. Il ne s'agit plus d'un polynôme et le modèle RBF n'est pas nécessairement quadratique. Il est construit par interpolation. Le modèle RBF permet de mieux représenter une fonction "compliquée" (fortement multimodale par exemple) que le modèle quadratique. Le modèle RBF est aussi une fonction assez "lisse" et robuste par rapport au bruit. Enfin, le modèle RBF est une fonction continue, deux fois différentiable [3] et dont les dérivées peuvent être calculées analytiquement.

L'utilisation de modèles RBF dans un algorithme de région de confiance a notamment été réalisée par Oeuvray dans sa thèse [25]. Il donne une raison

3. Ce n'est pas le cas pour toutes les fonctions à base radiale. Cependant, les fonctions gaussienne et multiquadrique implémentées dans Minamo vérifient ces propriétés.

supplémentaire d'utiliser un tel modèle. Lorsque les dérivées de la fonction objectif ne sont pas disponibles, le modèle doit être calculé par interpolation. Pour obtenir un modèle quadratique, $\frac{n(n+1)}{2} + n + 1$ points d'interpolations sont nécessaires, où n est le nombre de variables de la fonction. Un modèle RBF peut être calculé à partir d'un plus petit nombre de points. Il propose un ensemble d'interpolation de $n+1$ points minimum. Cette différence peut permettre un gain de temps, sachant que la fonction objectif est typiquement coûteuse à évaluer dans les problèmes rencontrés par Cenaero.

En pratique, dans notre cas, la construction du modèle RBF a demandé quelques ajustements. Nous avons implémenté une sorte de filtre qui permet de ne garder que des points "suffisamment différents[4]" pour construire le RBF. En effet, le calcul du modèle est impossible et donne lieu a un échec lorsque deux points identiques sont pris en compte. De plus, cela n'a aucun sens. Un tel filtre n'est pas nécessaire lorsqu'on utilise un algorithme génétique car l'aléatoire rend quasi nulle la probabilité d'ajouter deux fois le même point à la base de données. Cependant, avec les méthodes locales, il peut arriver un moment où les itérés deviennent fort proches. On peut ainsi obtenir une base de données contenant des points trop similaires pour pouvoir construire un RBF de façon adéquate.

Outre ce filtre appliqué automatiquement, l'utilisateur a le choix entre deux ensembles de points pour construire le RBF. La première façon consiste à construire le RBF à partir de tous les points "suffisamment différents" calculés jusqu'alors. La seconde manière ne sélectionne qu'un minimum de $n+1$ points situés dans un voisinage de la région de confiance. Le processus de sélection est initié à partir de la région de confiance qui est successivement élargie jusqu'à obtenir un minimum de $n+1$ points. Si plus de $n+1$ points sont situés dans la région de confiance, ils sont tous pris en compte. Cette seconde façon de procéder permet de construire un modèle qui est plus précis dans la région de confiance puisqu'il ne tient pas compte de l'information qui en est éloignée.

Nous avons construit deux méthodes de région de confiance. L'une utilise un modèle quadratique tandis que l'autre se base sur un modèle RBF. Elles sont respectivement présentées aux Sections 4.1.5 et 4.1.6.

4.1.3 La résolution du sous-problème

Le but de cette section est de présenter les méthodes qui permettent de calculer le pas qui "décroît suffisamment" le modèle tout en restant dans la région de confiance.

Dans un premier temps, nous avons considéré une procédure de backtracking permettant de calculer le point de Cauchy approximé pour déterminer le pas. Le point de Cauchy approximé à l'étape k, noté x_k^{AC}, est le point défini par

$$x_k^{AC} = x_k(j_c) = x_k - c_1^{j_c} \frac{\Delta_k}{\|g_k\|} g_k \tag{4.4}$$

4. Cette différence est mesurée par la valeur d'un paramètre, qui sera discutée au Chapitre 5.

et qui vérifie la condition

$$m_k(x_k(j_c)) \leq m_k(x_k) + c_2\langle g_k, x_k(j_c) - x_k\rangle,$$

où g_k représente le gradient du modèle à l'étape k, $c_1 \in]0,1[$ et $c_2 \in]0,\frac{1}{2}[$ sont des constantes données. Le processus consistait à minimiser le modèle à l'intérieur de la région de confiance et dans la direction de la plus forte pente (en procédant donc par backtracking). Le pas ainsi calculé permettait en effet d'assurer une décroissance suffisante du modèle pour assurer la convergence de la méthode de région de confiance d'un point de vue théorique (cfr. [5], Chapitre 6). Nous avons choisi de tester cette méthode car elle est utilisée dans la thèse de Oeuvray [25] et elle est relativement simple à implémenter. Cependant, en pratique la décroissance produite par cette méthode s'est révélée être trop faible et ralentir voire rendre inefficace la méthode de région de confiance. C'est pourquoi cette méthode n'a plus été considérée dans la suite de ce travail.

Nous avons ensuite utilisé deux autres méthodes pour déterminer le pas. Nous les décrivons dans les deux sections suivantes. La première méthode décrite, l'algorithme du gradient conjugué tronqué de Steihaug-Toint, permet de déterminer le pas lorsque le modèle est quadratique. Dans le cas des modèles RBF, nous avons utilisé l'algorithme IPOPT que nous présentons ensuite.

4.1.3.1 L'algorithme du gradient conjugué tronqué de Steihaug-Toint

Nous considérons que la fonction est modélisée par une approximation quadratique. Déterminer le pas revient donc à résoudre le problème

$$\min_{s\in\mathbb{R}^n} m(s) = \langle g, s\rangle + \frac{1}{2}\langle s, Hs\rangle \quad \text{sous contrainte} \quad ||s|| \leq \Delta,$$

où $m(s)$ correspond au modèle défini par l'équation (4.3) pour lequel on ommet le terme indépendant et l'indice k et Δ représente le rayon de la région de confiance. L'algorithme du gradient conjugué tronqué de Steihaug-Toint (cfr. [5], Chapitre 7) est une méthode itérative qui est une "généralisation" de l'algorithme du gradient conjugué. Celui-ci permet de générer une suite de directions p_k qui sont conjuguées pour la matrice H, i.e., qui vérifient la propriété

$$\langle p_k, Hp_i\rangle = 0 \quad \text{pour } k \neq i.$$

Mais la méthode du gradient conjugué ne peut s'appliquer que lorsque la courbure $\langle p_k, Hp_k\rangle$ est strictement positive, p_k représentant la direction dans laquelle l'itéré suivant est recherché. Or, ce n'est pas toujours le cas pour la résolution du sous problème qui nous intéresse. De plus, la suite des itérés pourrait sortir de la région de confiance. La méthode de Steihaug-Toint permet de gérer ces deux cas particuliers.

Commençons par traiter le premier cas, celui où $\langle p_k, Hp_k\rangle \leq 0$. Cela correspond à un modèle (quadratique) concave dans la direction p_k, qui n'est donc pas borné inférieurement dans la direction $s_k + \alpha p_k$, où s_k représente

l'itéré courant, p_k la direction de recherche et $\alpha \in \mathbb{R}^+$. Le minimum se trouvera donc sur la frontière de la région de confiance et sera la racine positive de

$$||s_k + \alpha p_k|| = \Delta. \tag{4.5}$$

En pratique, pour calculer cette racine (notons-la σ_k), on utilise l'expression

$$\sigma_k = \frac{-\langle s_k, p_k\rangle + \sqrt{\langle s_k, p_k\rangle^2 + ||p_k||^2(\Delta^2 - ||s_k||^2)}}{||p_k||^2}. \tag{4.6}$$

En effet, calculer la racine de (4.5) revient à calculer la racine de l'expression

$$||s_k + \alpha p_k||^2 - \Delta^2 = ||s_k||^2 - \Delta^2 + 2\alpha\langle s_k, p_k\rangle + \alpha^2||p_k||^2 \tag{4.7}$$

qui est une équation du second degré en α. Or (4.6) est la racine de (4.7).

Analysons maintenant le cas où $\langle p_k, Hp_k\rangle$ est strictement positif. On peut appliquer la méthode du gradient conjugué mais il y a toujours le risque que la suite des itérés sorte de la région de confiance. Dans ce cas, on peut s'arrêter et choisir le pas qui se trouve sur la frontière de la région. On peut en effet prouver qu'une fois qu'on sort de la région de confiance les itérés suivants n'y rentrent plus (si $s_0 = 0$). Comme dans le cas précédent, le pas sera alors défini selon (4.5) et calculé via (4.6).

Si la suite des itérés reste toujours à l'intérieur de la région de confiance, le pas s_k est calculé par la méthode classique du gradient conjugué (cfr [5] ou [11]). L'Algorithme 4.2 regroupe ces différentes étapes. Elles apparaissent dans le même ordre que la présentation faite ci-dessus. La convergence est déclarée lorsque la norme du gradient, $\|g_k\|$, est suffisamment proche de zéro.

Algorithme 4.2. (Gradient conjugué tronqué de Steihaug-Toint)
Poser $s_0 = 0$, $g_0 = g$ et $p_0 = -g_0$. Pour $k = 0, 1, ...$ jusqu'à convergence, réaliser les itérations :

Poser $\kappa_k = \langle p_k, Hp_k\rangle$.

Si $\kappa_k \leq 0$,

calculer σ_k comme la racine carrée positive de $||s_k+\sigma p_k|| = \Delta$,
poser $s_{k+1} = s_k + \sigma_k p_k$, et
stop.

Fin

Poser $\alpha_k = \langle g_k, g_k\rangle/\kappa_k$.

Si $||s_k + \alpha_k p_k|| \geq \Delta$,

calculer σ_k comme la racine carrée positive de $||s_k+\sigma p_k|| = \Delta$,
poser $s_{k+1} = s_k + \sigma_k p_k$, et
stop.

Fin

$$\begin{aligned} \text{Poser } s_{k+1} &= s_k + \alpha_k p_k, \\ g_{k+1} &= g_k + \alpha_k H p_k, \\ \beta_k &= \langle g_{k+1}, g_{k+1}\rangle/\langle g_k, g_k\rangle, \text{ et} \\ p_{k+1} &= -g_{k+1} + \beta_k p_k. \end{aligned}$$

Le pas calculé par l'Algorithme 4.2 permet d'obtenir une décroissance suffisante du modèle afin d'assurer la convergence de la méthode de région de confiance vers un point critique du premier ordre. En effet, le premier itéré de Steihaug-Toint, s_1, correspond au point de Cauchy approximé défini par l'équation (4.4). Les itérations suivantes permettent de tenir compte de l'information du second ordre et de décroître encore davantage la valeur du modèle (cfr [5], Chapitre 7, Section 5).

4.1.3.2 L'algorithme IPOPT

Nous considérons maintenant le problème

$$\min_{s \in \mathbb{R}^n} m_k(s) \quad \text{sous contrainte} \quad ||s - x_k|| \leq \Delta_k, \tag{4.8}$$

où $m_k(s)$ représente le modèle RBF (à l'itération k), x_k l'itéré courant et Δ le rayon de la région de confiance. Il s'agit d'un problème de minimisation non quadratique avec une contrainte d'inégalité.

Pour résoudre le problème (4.8), nous utilisons un algorithme de points intérieurs primal-dual avec une méthode de recherche linéaire et filtre. Celui-ci a été implémenté dans le code IPOPT par Carl Laird et Andreas Wächter. C'est la release 3.9.0, implémentée en C++ que nous avons utilisée [37]. L'algorithme ainsi que différents tests numériques sont présentés dans l'article [32]. Nous avons utilisé un code existant car notre but n'est pas de ré-implémenter des méthodes qui existent déjà mais de construire des méthodes hybrides et de mesurer leurs performances.

Deux éléments justifient principalement le choix de l'algorithme IPOPT pour résoudre le sous-problème de la région de confiance. D'une part, IPOPT est utilisé dans l'algorithme DFO, qui est une des méthodes locales qui intervient dans notre algorithme hybride (cfr. Section 4.2). D'autre part, le code IPOPT est disponible en C++ qui est le langage dans lequel est codé Minamo. L'appel au code IPOPT depuis le logiciel de Cenaero en est ainsi facilité. Signalons toutefois que la mise en place de l'interface n'a pas été si évidente. Une première manipulation du code IPOPT a d'abord été effectuée, de façon indépendante à Minamo afin de bien en saisir le fonctionnement. Ensuite, lors de l'utilisation à l'intérieur de Minamo, différentes difficultés sont apparues. Des modifications ont dû être apportées dans différentes classes déjà implémentées dans Minamo afin de permettre une utilisation correcte du code IPOPT. Citons par exemple l'implémentation de procédures de clonage pour les classes faisant intervenir des pointeurs. Finalement, remarquons qu'IPOPT est peut être une méthode trop complète pour résoudre ce problème. Une méthode de type gradient conjugué projeté aurait pu être utilisée.

4.1.4 La mise à jour du rayon

En ce qui concerne la mise à jour du rayon de la région de confiance, différentes possibilités existent. Nous en avons retenu deux. La première méthode est la plus classique. Elle consiste à multiplier le rayon par deux lorsque

l'itération est très réussie ($\rho_k \geq \eta_2$) et à le diviser par deux lorsque l'itération n'est pas réussie, c'est-à-dire $\rho_k < \eta_1$. Si l'itération est réussie mais que $\rho_k < \eta_2$, on conserve le même rayon pour l'itération suivante. On applique donc la règle

$$\Delta_{k+1} = \begin{cases} 2\Delta_k & \text{si } \rho_k \geq \eta_2, \\ \Delta_k & \text{si } \rho_k \in [\eta_1, \eta_2[, \\ \frac{1}{2}\Delta_k & \text{si } \rho_k < \eta_1, \end{cases}$$

qui respecte les conditions de la cinquième étape de l'Algorithme 4.1. La seconde méthode fait intervenir la longueur du pas, $\|s_k\|$, calculée au cours de la dernière itération. Si l'itération est très réussie ($\rho_k \geq \eta_2$), le nouveau rayon est le maximum entre le rayon actuel et la norme du pas multipliée par un coefficient supérieur à 1. Si $\rho_k < \eta_1$, le nouveau rayon est égal à une fraction de la norme du pas. Si l'itération est réussie mais $\rho_k < \eta_2$, le rayon est inchangé. La règle appliquée est donc la suivante :

$$\Delta_{k+1} = \begin{cases} \max\{\alpha_1 ||s_k||, \Delta_k\} & \text{si } \rho_k \geq \eta_2, \\ \Delta_k & \text{si } \rho_k \in [\eta_1, \eta_2[, \\ \alpha_2 ||s_k|| & \text{si } \rho_k < \eta_1, \end{cases}$$

où α_1 et α_2 sont des constantes qui vérifient $0 < \alpha_2 < 1 \leq \alpha_1$. Dans le cas où $\rho_k < \eta_1$, cette seconde méthode permet de réduire plus rapidement le rayon de la région de confiance que lorsqu'il est divisé par deux à chaque itération. On réduit ainsi le nombre d'itérations nécessaires pour obtenir un coefficient ρ_k satisfaisant, ce qui permet d'épargner des évaluations de la fonction objectif qui peuvent être fort coûteuses. Pour plus de détails sur ce processus, on pourra se rapporter à la Section 10.5.2 de [5].

4.1.5 La première méthode : modèle quadratique

Cette première implémentation de l'Algorithme 4.1 fait intervenir un modèle RBF de la fonction objectif définie par le problème (4.1) et un modèle quadratique du RBF que nous venons de citer. Comme pour la phase d'optimisation globale où l'algorithme génétique optimise un modèle RBF de la fonction objectif exacte, notre méthode de région de confiance va tenter de déterminer l'optimum de ce modèle RBF. L'expression analytique du modèle RBF et de ses dérivées est connue. Il est donc facile d'en construire un modèle quadratique donné par l'équation (4.3), en posant g_k et H_k comme étant respectivement le gradient et le hessien du RBF.

Dans un premier temps, nous avons réalisé une implémentation assez basique. La première étape consiste à déterminer la fonction objectif qui sera minimisée par l'algorithme de région de confiance. Pour ce faire, on construit un modèle RBF global à partir de tous les points où la valeur de la fonction exacte est connue, i.e., les points du DoE ainsi que ceux évalués durant l'algorithme génétique. L'Algorithme 4.1 de région de confiance est ensuite utilisé pour déterminer l'optimum du RBF. Rappelons qu'un modèle quadratique de ce RBF est utilisé au cours de l'algorithme et que la minimisation de ce dernier à l'intérieur de la région de confiance est réalisée via la méthode de Steihaug-Toint. Le schéma de la méthode locale ainsi implémentée est représenté à la Figure 4.1.

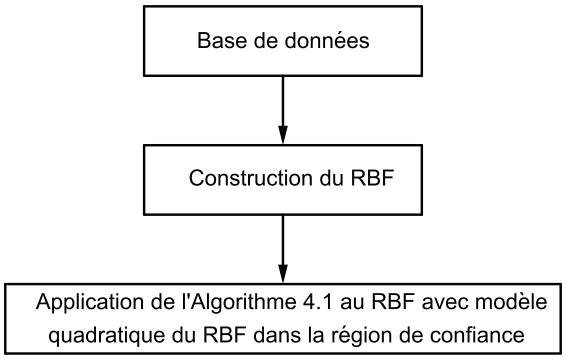


FIGURE 4.1 – Schéma général de la première implémentation de la région de confiance appliquée sur le RBF avec modèle quadratique.

Une telle implémentation permet de déterminer l'optimum du RBF. Cependant, celui-ci peut être différent du point qui minimise la fonction objectif exacte. De plus, aucun appel à la fonction objectif n'est effectué au cours de la phase locale. En effet, la fonction exacte n'intervient que lors de la construction du RBF. Il semble donc judicieux de faire intervenir cette évaluation exacte à certains moments de la stratégie d'optimisation locale afin d'augmenter les chances de convergence vers un minimum de la vraie fonction objectif. Ceci nous amène à considérer de nouvelles implémentations basées sur l'Algorithme 4.1.

La seconde implémentation envisagée est une façon naïve de faire appel à la fonction objectif exacte. A la fin de chaque itération de l'Algorithme 4.1, le nouvel itéré calculé est évalué via la fonction exacte. Pour tenir compte de cette nouvelle information disponible, on ajoute ce point à la base de données utilisée pour la construction du RBF. On obtient ainsi un RBF susceptible d'être plus proche de la vraie fonction à chaque itération. Cependant, on réalise que cette démarche n'est pas correcte par rapport à la méthode de région de confiance. En effet, le RBF correspond à la fonction minimisée par l'algorithme de région de confiance. Or cette fonction est supposée rester identique durant toute la durée de l'algorithme, ce qui n'est plus le cas lorsqu'on construit le RBF avec davantage de points à chaque itération puisque le RBF est déterminé par les points à partir desquels il est construit.

Finalement, nous avons choisi d'implémenter la méthode de région de confiance avec modèle quadratique du RBF suivant un schéma similaire à celui de l'algorithme génétique utilisé dans la phase globale (cfr. Section 3.4). Le principe est d'imbriquer deux boucles. Une itération de la boucle externe comprend la construction d'un modèle RBF à partir de l'information connue jusqu'alors. L'Algorithme 4.1 est alors appliqué sur ce modèle RBF. La solution ainsi obtenue est évaluée via la simulation exacte et ajoutée à la base de données. On peut ensuite passer à l'itération suivante. Une itération externe a donc un coût d'une seule évaluation de la fonction objectif exacte. Les itérations internes correspondent aux itérations de l'Algorithme 4.1 (indicées par k). Le schéma final de notre première méthode locale est synthétisé à la

Figure 4.2.

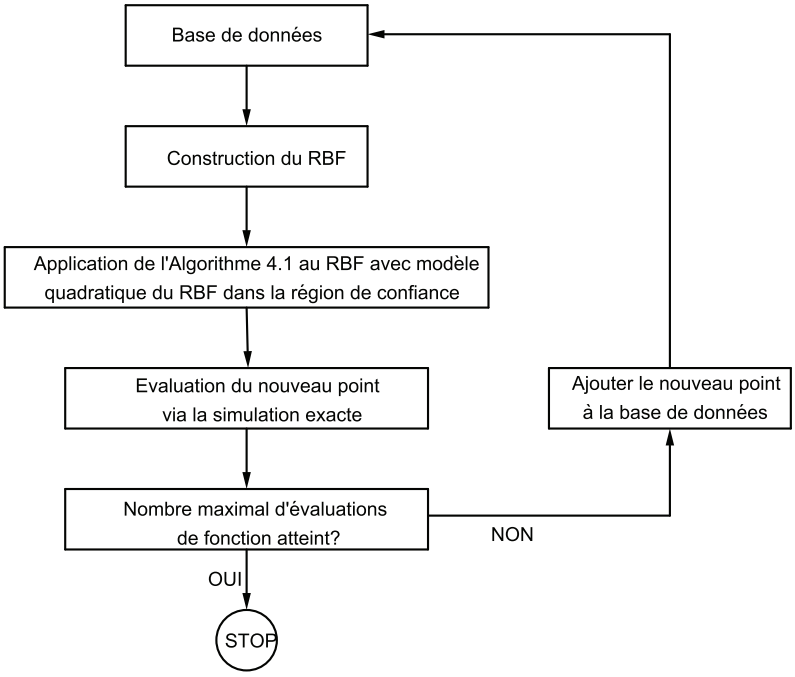


FIGURE 4.2 – Schéma général de l'implémentation finale de la région de confiance appliquée sur le RBF avec modèle quadratique.

Un dernier point important à discuter avant de clôturer la section sur cette première méthode locale concerne les critères d'arrêt. Le nombre d'itérations de la boucle externe correspond au nombre d'évaluations de fonction utilisées par la méthode locale. Le seul critère d'arrêt que nous utilisons est un nombre maximal d'itérations (et donc indirectement d'évaluations de fonction). Des critères supplémentaires (prenant en compte la norme du gradient du RBF par exemple) pourront être pris en compte par la suite mais l'objectif de ce mémoire est d'analyser les performances de l'algorithme pour un nombre donné d'évaluations de fonction. C'est pourquoi nous nous limitons à ce seul critère pour la boucle externe. Au niveau des itérations internes, le critère d'arrêt correspond au critère d'arrêt de l'Algorithme 4.1. Le critère classique pour un tel algorithme est de considérer une borne inférieure sur la norme du gradient (du RBF dans le cas qui nous préoccupe). Cependant ce critère s'est révélé trop faible dans certains cas car il donnait lieu à des itérations inutiles qui n'apportaient plus d'amélioration à la solution calculée. C'est pourquoi nous avons utilisé deux autres critères en nous basant sur le livre de Dennis et Schnabel [7]. Le premier critère fait intervenir le gradient mais de façon relative. Il est donné par

$$\max_{1\leq i\leq n}\left\{\frac{|\nabla f(x_k+s)_i|\max\{|x_{k_i}|,typx_i\}}{\max\{|f|,typf\}}\right\}\leq gradtol,$$

où $typx$ est un vecteur qui donne l'ordre de grandeur des variables x, $typf$ un nombre positif qui correspond à l'ordre de grandeur de la fonction f autour de

la solution et *gradtol* un paramètre qui correspond à la précision souhaitée. Le second critère permet d'arrêter l'algorithme lorsque le pas calculé devient trop petit que pour encore pouvoir apporter une amélioration de la solution. Il est donné par

$$\max_{1\leq i\leq n}\left\{\frac{|(x_k+s)_i - x_{k_i}|}{\max\{|(x_k+s)_i|, typx_i\}}\right\} \leq steptol,$$

où *steptol* est un paramètre qui correspond à la distance entre deux itérés successifs à partir de laquelle l'algorithme est terminé. Finalement, nous avons ajouté un critère sur un nombre maximal d'itérations internes. Le choix des différents paramètres présentés juste avant sera discuté au Chapitre 5.

4.1.6 La seconde méthode : modèle RBF

L'idée de cette seconde méthode locale est de construire une méthode de région de confiance qu'on applique directement sur la fonction objectif exacte. Comme on travaille avec des fonctions qui peuvent être fortement multimodales et dont on ne connaît pas nécessairement l'expression des dérivées, on choisit de modéliser cette fonction à l'intérieur de la région de confiance par un modèle RBF.

Le schéma de cette méthode est donc simple et classique puisqu'il reprend les différentes étapes de l'Algorithme 4.1. Comme la fonction optimisée par la méthode de région de confiance est la fonction objectif exacte, un nouveau point est évalué via cette dernière à chaque itération. Le modèle m_k pouvant varier avec les itérations, nous pouvons cette fois construire un modèle RBF qui varie au cours des itérations, en prenant en compte la nouvelle information disponible. Le but est d'obtenir un modèle RBF qui approxime de mieux en mieux la fonction exacte au plus les itérations avancent. Le modèle RBF n'étant pas quadratique, le sous-problème de la région de confiance est résolu via IPOPT. La Figure 4.3 reprend un schéma des différentes étapes de cette seconde méthode de région de confiance.

Une telle implémentation de l'Algorithme 4.1 possède un coût d'une évaluation de la fonction objectif par itération. Nous avons choisi le critère d'arrêt dans le même esprit que pour la première méthode de région de confiance. Nous voulons observer les performances de notre méthode pour un nombre maximal donné d'évaluations de la fonction objectif. C'est pourquoi le critère d'arrêt retenu est un nombre maximal d'itérations, qui est équivalent à un nombre maximal d'évaluations de fonction au regard du coût d'une itération.

Finalement, nous terminons cette partie sur nos méthodes de région de confiance par une remarque sur leur implémentation. Le schéma de la seconde méthode est plus simple que celui de la première. Pourtant, au niveau de l'implémentation, coder la seconde a nécessité plus de temps et de travail. Une évaluation de la fonction objectif exacte intervient à l'intérieur d'une itération. Il a donc fallu trouver des astuces pour réaliser cette évaluation car Minamo est conçu de telle sorte que l'évaluation exacte soit effectuée en fin d'itération.

Nous allons maintenant présenter dans les deux sections suivantes les deux autres méthodes locales que nous utilisons dans nos couplages. Il s'agit

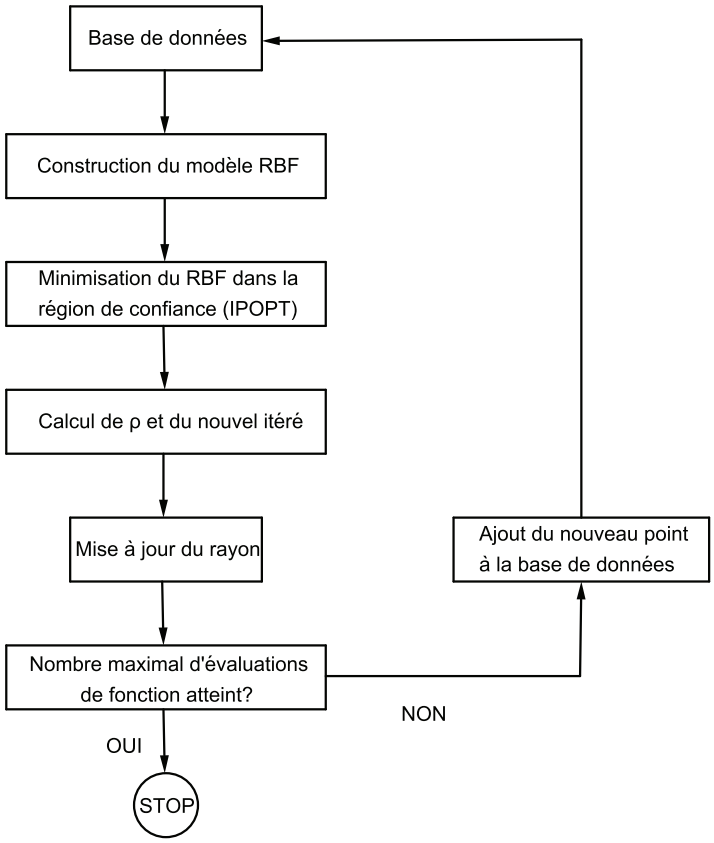


FIGURE 4.3 – Schéma général de l'implémentation de la région de confiance appliquée sur la fonction exacte avec modèle RBF.

de méthodes que nous n'avons pas implémentées, nous avons utilisé des codes existants.

4.2 L'algorithme DFO

L'algorithme DFO, pour Derivative Free Optimization, a été développé [5] dans le cadre de l'optimisation de fonctions "lisses" mais non linéaires, à plusieurs variables et dont les dérivées ne sont pas disponibles. L'évaluation de ces fonctions peut également être très coûteuse. La construction de ce nouvel algorithme a été motivée par une demande croissante pour de tels outils. Il existe peu de stratégies pour résoudre ce type de problèmes. Conn et al. en citent 4 dans leurs articles [17] et [18] : la recherche directe, les outils de différentiation automatique, l'utilisation de différences finies pour approximer le gradient et le hessien (ou une approximation quasi-Newton du hessien) et la modélisation de la fonction objectif plutôt que celle des dérivées. C'est cette dernière approche qui a été retenue et est au coeur de l'algorithme DFO. Il s'agit d'un algorithme de type région de confiance basé

5. Le code que nous utiliserons pour cet algorithme a été implémenté par K. Scheinberg.

sur un modèle quadratique de la fonction objectif. Le modèle est construit sans faire intervenir le gradient ou le hessien de la fonction objectif, c'est une méthode d'interpolation qui est appliquée. Nous commençons par décrire la construction de ce modèle avant de présenter les différentes étapes de l'algorithme. Cette section se base sur les articles [17, 18, 19] et le livre [6].

4.2.1 La construcion du modèle

A chaque itération de l'algorithme, la fonction objectif $f(x_k)$ est approchée par un modèle quadratique $m_k(x_k + s)$ de la forme

$$m_k(x_k + s) = f(x_k) + \langle g_k, s\rangle + \frac{1}{2}\langle s, H_k s\rangle,$$

où x_k représente l'itéré courant, g_k un vecteur de $\mathbb{R}^n$ et H_k une matrice symétrique de $\mathbb{R}^n$ si n représente la dimension du problème. La construction de ce modèle revient à déterminer g_k et H_k. On choisit de réaliser une construction par interpolation. L'ensemble d'interpolation Y contient les points utilisés pour bâtir le modèle, à partir de conditions de la forme

$$m(y) = f(y), \quad \forall y \in Y. \tag{4.9}$$

Deux aspects interviennent dans la construction de l'interpolation : le choix des points d'interpolation et le choix des polynômes de base qui seront combinés pour obtenir le modèle.

La détermination des points d'interpolation est influencée par le nombre de points utilisés ainsi que par leur disposition dans l'espace. Pour obtenir un modèle *complètement quadratique*, l'ensemble Y doit contenir $p = \frac{(n+1)(n+2)}{2}$ points. Cette condition est nécessaire pour que les éléments g_k et H_k soient déterminés de façon unique. Cependant, un modèle construit sur un plus petit nombre de points peut être suffisamment précis et permettre de travailler de façon acceptable. L'idée proposée est de commencer l'algorithme avec un faible nombre de points appartenant à Y et d'augmenter ce nombre au cours des itérations (grâce aux points évalués durant les différentes étapes de l'algorithme). Pour assurer une précision suffisante du modèle et une bonne progression de l'algorithme, un modèle *complètement linéaire* est requis, i.e., construit à partir de $n + 1$ points au moins.[6]

Lorsque le problème est multidimensionnel ($n > 1$), un modèle construit à partir de $p = \frac{(n+1)(n+2)}{2}$ points n'est pas nécessairement unique. Une condition sur la disposition des points dans l'espace doit également être vérifiée pour assurer l'existence et l'unicité du modèle. Cette condition porte le nom de *poisdness*. Une façon d'exprimer qu'un ensemble de points $Y = \{y_1, ..., y_p\}$ vérifie cette condition est d'utiliser une base $\{\phi_i(.)\}_{i=1}^p$ de l'espace linéaire des quadratiques à n dimensions. Si le déterminant

$$\delta(Y) = \det \begin{pmatrix} \phi_1(y_1) & ... & \phi_1(y_p) \\ \vdots & \ddots & \vdots \\ \phi_p(y_1) & ... & \phi_p(y_p) \end{pmatrix}$$

6. C'est aussi le nombre de points que nous avons choisi d'utiliser dans nos méthodes locales.

est non nul, l'ensemble Y sera dit *poised*.

Dans l'algorithme, la détermination d'un ensemble Y *poised* ne se fait pas via le calcul du déterminant présenté ci-dessus. Elle est liée au choix des polynômes de base. Le modèle quadratique de l'algorithme est une interpolation basée sur des polynômes fondamentaux de Newton. Ce choix est motivé par des raisons théoriques et pratiques. L'utilisation des polynômes de Newton permet d'établir une théorie de convergence globale de l'algorithme (cfr. [18]) et de réduire les calculs effectués par l'algorithme. Lorsque l'ensemble d'interpolation est *poised*, le modèle obtenu est toujours le même, quelle que soit la méthode d'interpolation choisie. L'avantage d'une interpolation basée sur les polynômes de Newton est de permettre la création de procédures pour améliorer la géométrie de l'ensemble d'interpolation et de méthodes pour inclure de nouveaux points dans Y.

Le principe de l'interpolation basée sur les polynômes fondamentaux de Newton est présenté dans les articles [17] et [18] ainsi que dans le livre [5]. La méthode décrit la construction d'un interpolant de degré d. L'ensemble d'interpolation Y est séparé en $d+1$ blocs $Y^{[l]}$, où le bloc d'indice l contient

$$\begin{pmatrix} l+n-1 \\ l \end{pmatrix} = \frac{(l+n-1)!}{l!\,(n-1)!}$$

points (pour $l = 0, ..., d$). On fait correspondre à chaque point $y_i^{[l]} \in Y^{[l]}$ un unique polynôme fondamental de Newton de degré l qui vérifie la condition

$$N_i^{[l]}(y_j^{[m]}) = \delta_{ij}\delta_{lm} \text{ pour tout } y_j^{[m]} \in Y^{[m]} \text{ avec } m \leq l.$$

Le polynôme d'interpolation est obtenu en prenant une combinaison linéaire des polynômes fondamentaux

$$m(x) = \sum_{y_i^{[l]} \in Y} \lambda_i^{[l]}(Y, f) N_i^{[l]}(x),$$

où les coefficients $\lambda_i^{[l]}(Y, f)$ sont des différences finies généralisées appliquées à la fonction f. Pour construire les polynômes fondamentaux, on utilise la procédure décrite par l'Algorithme 4.3.

Algorithme 4.3. (Construction des polynômes)
Initialiser les $N_i^{[l]}$ pour $i = 1, ..., |Y^{[l]}$, et $l = 0, ..., d$ à la base de polynômes choisie (les monômes par exemple).
Poser $Y_{temp} = \emptyset$.
Pour $l = 0, ..., d$,
  pour $i = 1, .., |Y^{[l]}|$
    choisir $y_i^{[l]} \in Y \backslash Y_{temp}$ tel que $|N_i^{[l]}(y_i^{[l]})| \neq 0$,
    si un tel $y_i^{[l]}$ n'existe pas, réinitialiser $Y = Y_{temp}$ et STOP
    (la base des polynômes de Newton est incomplète),
    $Y_{temp} \leftarrow Y_{temp} \cup \{y_i^{[l]}\}$
    normaliser le polynôme courant par

$$N_i^{[l]}(x) \leftarrow N_i^{[l]}(x)/|N_i^{[l]}(y_i^{[l]})|, \tag{4.10}$$

mettre à jour tous les polynômes de Newton du bloc l et des blocs suivants par

$$N_j^{[l]}(x) \leftarrow N_j^{[l]}(x) - N_j^{[l]}(y_i^{[l]})N_i^{[l]}(x) \quad j \neq i,\ j = 1, ..., |Y^{[l]}|,$$

$$N_j^{[k]}(x) \leftarrow N_j^{[k]}(x) - N_j^{[k]}(y_i^{[l]})N_i^{[l]}(x) \quad j = 1, ..., |Y^{[k]}|,\ k = l+1, ..., d.$$

fin

Fin (la base des polynômes de Newton est complète).

Les dénominateurs de l'équation (4.10) sont appelés les *pivots*. Si ceux-ci sont non nuls, la propriété *poisedness* est assurée. L'Algorithme 4.3 permet donc d'obtenir un ensemble Y qui vérifie cette propriété. Cependant, on sera plus exigeant en imposant de travailler avec un ensemble *well-poised*, i.e., les pivots doivent être non nuls et suffisamment grands. On impose donc la condition $|N_i^{[l]}(y_i^{[l]})| > \theta$ pour $\theta > 0$ au lieu de $|N_i^{[l]}(y_i^{[l]})| \neq 0$. Ce nouveau paramètre, appelé *pivot threshold*, apparaîtra dans les options du code de l'algorithme DFO.

4.2.2 L'algorithme

L'algorithme DFO est principalement basé sur une méthode de région de confiance. La fonction objectif f est modélisée par un modèle quadratique. A chaque itération, celui-ci est construit via une interpolation basée sur les polynômes fondamentaux de Newton, processus décrit à la section précédente. Le modèle est considéré comme suffisamment précis dans la région de confiance $\mathcal{B}_k$, de rayon Δ_k. On note x_k l'itéré courant et s_k le pas calculé à l'itération k. Les constantes

$$0 < \eta_0 \leq \eta_1 < 1, \quad 0 < \gamma_0 \leq \gamma_1 < 1 \leq \gamma_2$$

interviennent dans les différentes étapes de l'Algorithme 4.4, issu de l'article [17].

Algorithme 4.4. (Derivative free trust region algorithm)

1. Initialisations.
   Soit x_s et $f(x_s)$. Choisir un ensemble d'interpolation initial Y contenant x_s et au moins un autre point. Déterminer $x_0 \in Y$ tel que $f(x_0) = \min_{y_i \in Y} f(y_i)$. Choisir un rayon initial $\Delta_0 > 0$. Poser $k = 0$.
2. Construction du modèle.
   En utilisant l'ensemble d'interpolation Y, construire un modèle $m_k(x_k + s_k)$, en restreignant éventuellement Y à un sous-ensemble *poised* et contenant x_k, de telle sorte que les conditions (4.9) soient valables pour l'ensemble Y résultant.
3. Minimiser le modèle dans la région de confiance.
   Calculer le point $x_k + s_k$ tel que

   $$m_k(x_k + s_k) = \min_{x \in \mathcal{B}_k} m_k(x).$$

   Calculer $f(x_k + s_k)$ et le rapport

   $$\rho_k \stackrel{def}{=} \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}.$$

4. Mettre à jour l'ensemble d'interpolation
   - Si $\rho_k \geq \eta_1$, introduire $x_k + s_k$ dans Y. Si $|Y| = p$ [7], retirer un des points d'interpolation déja présent dans Y.
   - Si $\rho_k < \eta_1$ et Y n'est pas *adéquat* dans $\mathcal{B}_k$, *améliorer* la géométrie dans $\mathcal{B}_k$.
5. Mettre à jour le rayon.
   - Si $\rho_k \geq \eta_1$, poser
   $$\Delta_{k+1} \in [\Delta_k, \gamma_2 \Delta_k].$$
   - Si $\rho_k < \eta_1$ et Y n'est pas adéquat dans $\mathcal{B}_k$, poser
   $$\Delta_{k+1} \in [\gamma_0 \Delta_k, \gamma_1 \Delta_k].$$
   - Sinon, poser $\Delta_{k+1} = \Delta_k$.
6. Mettre à jour l'itéré courant.
   Déterminer $\hat{x}_k$ tel que
   $$f(\hat{x}_k) = \min_{y_i \in Y \text{ et } y_i \neq x_k} f(y_i).$$
   Alors, si
   $$\hat{\rho}_k \stackrel{def}{=} \frac{f(x_k) - f(\hat{x}_k)}{m_k(x_k) - m_k(x_k + s_k)} \geq \eta_0,$$
   poser $x_{k+1} = \hat{x}_k$. Sinon, poser $x_{k+1} = x_k$. Incrémenter k de 1 et retourner à l'étape 1.

La principale différence par rapport à un algorithme de région de confiance classique réside dans la quatrième étape. On y voit également apparaître deux nouveaux concepts : un ensemble adéquat de points et l'amélioration de sa géométrie. La notion d'ensemble adéquat fait intervenir une borne supérieure sur les polynômes fondamentaux de Newton et impose que l'ensemble contienne au moins $n+1$ points. La phase d'amélioration a pour but de rendre adéquat un ensemble qui ne l'est pas. Cette étape est nécessaire car l'ajout d'un nouveau point à l'ensemble peut détériorer sa géométrie. Pour plus de détails sur ces procédures, on pourra se rapporter aux articles [17, 18].

Une théorie de convergence a été développée pour l'algorithme DFO. La convergence globale de l'algorithme vers des points critiques du premier ordre a été démontrée dans l'article [18]. Pour plus de détails sur l'algorithme, on pourra également se référer à l'article [17].

4.2.3 Le code utilisé

Dans notre algorithme hybride, nous avons testé le couplage de l'algorithme génétique avec cette méthode DFO. Nous n'avons pas implémenté cet algorithme. Nous avons utilisé le code DFO [38], implémenté par K. Scheinberg (entre 1998 et 2003), qui implémente l'algorithme que nous venons de

7. La valeur p représente la taille maximale de l'ensemble d'interpolation.

présenter. L'article [19] présente l'aspect pratique de l'algorithme DFO ainsi que quelques résultats numériques.

DFO est un code écrit en FORTRAN77 et qui permet l'optimisation de fonctions coûteuses, lisses et dont les dérivées ne sont pas disponibles. DFO est conseillé pour les problèmes de dimension inférieure à 50. Au delà de cette valeur, l'exécution du programme peut devenir très lente, même pour des fonctions peu coûteuses. Pour résoudre le sous-problème de minimisation du modèle dans la région de confiance (étape 3 de l'Algorithme 4.4), DFO fait appel à la version FORTRAN77 du package IPOPT présenté à la Section 4.1.3.2.

Comme DFO a été développé pour optimiser des fonctions coûteuses, il permet de prendre en compte toute l'information déja disponible sur la fonction avant le lancement de l'algorithme. En effet, si certains points ont déja été évalués via la fonction objectif lors d'un autre processus, il est possible de passer l'ensemble de ces points ainsi que leurs valeurs à l'algorithme. DFO choisira alors son point initial parmi cet ensemble de candidats, en retenant celui qui possède la plus petite valeur de la fonction objectif. Cela peut permettre un gain de temps considérable.

Outre le point initial (ou l'ensemble initial de points) et la dimension du problème, d'autres paramètres sont également passés à l'algorithme. Un nombre maximal d'itérations est indiqué mais aussi un nombre maximal d'évaluations de la fonction objectif, critère d'arrêt souvent utilisé lors de l'optimisation de fonctions coûteuses. Le principal critère d'arrêt de DFO reste fixé par le rayon minimal de la région de confiance. Une fois cette valeur atteinte, l'algorithme se termine. Il est possible de terminer l'algorithme plus rapidement en activant l'option qui permet de quitter l'algorithme si peu de progrès a été réalisé durant deux itérations consécutives. Le choix des valeurs des différents paramètres sera explicité dans le chapitre consacré aux tests et résultats numériques.

Le couplage avec l'algorithme génétique de Minamo a été réalisé de façon externe. L'algorithme DFO est lancé une fois l'exécution de Minamo terminée et n'est pas inclu dans le logiciel. Les points évalués dans Minamo et leur valeur évaluée par la fonction objectif, ainsi que l'optimum déterminé par l'algorithme génétique sont transmis à DFO via des fichiers. Pour évaluer la fonction objectif, DFO fait appel à la simulation exacte (la même que celle utilisée par l'algorithme génétique) grâce à un appel système. Les résultats obtenus par cette méthode hybride sont présentés au Chapitre 5.

4.3 L'algorithme BCDFO

L'algorithme BCDFO, pour Bound Constrained Derivative Free Optimization, est une méthode de région de confiance avec *active-set*, pour des problèmes d'optimisation non linéaires, sans dérivée et avec contraintes de bornes. Cet algorithme a été développé et implémenté tout récemment par Anke Tröltzsch, Serge Gratton et Philippe L. Toint. La description que nous réalisons ici est basée sur l'article [23]. L'algorithme permet de résoudre le

problème d'optimisation avec les contraintes de bornes

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x), \\ \text{sous contrainte} \quad & l \leq x \leq u, \end{aligned} \tag{4.11}$$

où f est une fonction non linéaire de $\mathbb{R}^n$ dans $\mathbb{R}$ bornée inférieurement et l et u sont les vecteurs de bornes inférieures et supérieures sur x.

La méthode employée pour résoudre (4.11) est de type région de confiance. La fonction objectif est modélisée par une approximation quadratique, obtenue par une interpolation qui utilise des polynômes de Lagrange. Une partie importante de l'algorithme est celle consacrée à la construction d'un ensemble de points d'interpolation qui conviennent. On retrouve le problème d'ensemble *well-poised* présenté à la section consacrée à l'algorithme DFO. La minimisation du modèle à l'intérieur de la région de confiance est réalisée grâce à un algorithme de gradient conjugué projeté tronqué pour tenir compte des contraintes de bornes [4].

Les contraintes de bornes sont gérées par une méthode d'*active-set*. Les contraintes actives et presqu'actives sont repérées. La minimisation est alors effectuée dans l'espace des variables encore libres. L'algorithme est récursif. L'Algorithme 4.5 reprend la description générale de l'algorithme BCDFO comme présentée dans l'article [23].

Algorithme 4.5. BCDFO$(S_0, X_0, x_0, Z_0, \Delta_0, \epsilon)$

Pas 0 : Initialiser.

Un rayon de la région de confiance Δ_0 et un seuil de précision ϵ sont donnés. X_0, l'ensemble de tous les points et un ensemble d'interpolation candidat Z_0 qui contient le point initial x_0 sont aussi donnés. L'ensemble S_0 qui représente l'espace de recherche est donné. Poser $k = 0$.

Pas 1 : Assurer la conformité de Z_0 et construire le modèle initial.

A partir de Z_0, construire l'ensemble d'interpolation Y_0 approprié pour construire un modèle d'interpolation avec $|Y_0| \geq \dim(S_0)+1$. Ensuite, construire le modèle d'interpolation correspondant, m_0.

Pas 2 : Restreindre (éventuellement) la minimisation à un sous-espace S_k.

Pas 2.1 : Contrôler des bornes (presque) actives.

Déterminer les bornes actives et presqu'actives, ainsi que le sous-espace correspondant S_k engendré par les variables toujours libres. S'il n'y a pas de contrainte active ou presqu'active ou si S_k a déjà été exploré, aller au Pas 3.

Pas 2.2 : Projeter l'information sur le sous-espace des variables libres.

Projeter les points de X_k qui sont proches des contraintes (presque) actives sur S_k et leur associer une estimation conforme de la valeur de fonction.

Pas 2.3 : Construire un ensemble d'interpolation candidat dans le sous-espace.

Construire un nouvel ensemble d'interpolation candidat Z_k dans S_k qui inclut les points projetés s'il en existe.

Pas 2.4 : Résoudre dans S_k par un appel récursif.
Appeler l'algorithme

$$BCDFO(S_k, X_k, x_k, Z_k, \Delta_k, \epsilon),$$

qui produit une solution x_S^* du problème dans le sous-espace.
Pas 2.5 : Retourner à l'espace complet.
Si $\dim(S_k) < n$, renvoyer x_S^*. Sinon, redéfinir $x_k = x_S^*$, construire un nouvel ensemble d'interpolation Y_k autour de x_k et construire le modèle correspondant m_k.

Pas 3 : Test de criticalité.
Si $\|\mathcal{P}_{\mathcal{F}}(x_k - \nabla m_k(x)) - x_k\|_\infty \leq \epsilon$ (où $\mathcal{P}_{\mathcal{F}}$ est la projection du gradient sur $\mathcal{F}$) et le modèle m_k est suffisamment précis, renvoyer x_k.

Pas 4 : Calculer un point candidat et évaluer la fonction objectif.
Calculer $x_k^+ = x_k + s_k$ en appliquant un algorithme de gradient conjugué projeté tronqué. Evaluer f en x_k^+ et calculer le ratio ρ_k via (4.2).

Pas 5 : Definir l'itéré suivant et mettre à jour le rayon de la région de confiance.
Décider de la manière éventuelle d'intégrer le candidat x_k^+ dans l'ensemble Y_{k+1}, définir x_{k+1} et déterminer Δ_{k+1}.

Pas 6 : Mettre à jour le modèle.
Si $Y_{k+1} \neq Y_k$, calculer le modèle d'interpolation m_{k+1} autour de x_{k+1} en utilisant Y_{k+1}. Mettre à jour $X_{k+1} = X_k \cup \{x_{k+1}\}$. Incrémenter k de 1 et aller au Pas 2.

Un appel initial classique à l'Algorithme 4.5 est réalisé avec $S_0 = \mathbb{R}^n$ et $X_0 = Z_0 = \{x_0\}$. Cependant, l'appel à BCDFO dans notre méthode hybride est légèrement différent. Nous ne passons pas uniquement un point initial x_0. Afin d'exploiter toute l'information accumulée durant la phase d'optimisation globale, nous passons à l'algorithme l'ensemble des points évalués précédemment ainsi que le résultat de leur évaluation via la fonction objectif exacte. Une telle démarche est similaire à celle réalisée pour l'appel de l'algorithme DFO. Nous obtiendrons ainsi des résultats qui peuvent être comparés.

En ce qui concerne l'implémentation, nous avons utilisé le code Matlab implémenté par Anke Tröltzsch, Serge Gratton et Philippe L. Toint. Il s'agit d'une version reçue le 21 mars 2011. N'ayant pas de licence Matlab à Cenaero, les tests ont été effectués sous la version 3.2.3 d'Octave. Dans la grande majorité des cas, l'algorithme a tourné correctement. Certains runs ont toutefois échoué, suite à une mauvaise gestion des variables globales par Octave. Dans le cas des échecs, nous avons relancé l'algorithme en utilisant Matlab.

4.4 Les algorithmes hybrides

Nous venons de présenter quatre méthodes locales, de type région de confiance. A la fin du Chapitre 3, nous avions un schéma pour nos méthodes hybrides (cfr. Figure 3.3) où le contenu de la phase locale restait inconnu. Nous pouvons maintenant compléter ce cadre grâce aux descriptions que nous venons de faire dans les sections précédentes. Nous disposons de quatre méthodes locales avec lesquelles nous allons construire quatre algorithmes hybrides.

Le premier algorithme hybride est obtenu en complétant le cadre de la phase globale par le schéma donné à la Figure 4.2. On obtient ainsi une méthode qui combine un algorithme génétique et une méthode de région de confiance appliquée à un modèle RBF de la fonction excate, modèle qui sera à son tour approximé par un modèle quadratique cette fois.

Lorsque la phase locale consiste à appliquer le schéma présenté à la Figure 4.3, on obtient une seconde méthode hybride. A la suite de la phase globale, cette approche applique la méthode de région de confiance sur la fonction objectif exacte. Le modèle utilisé dans la région de confiance sera un RBF.

La troisième méthode hybride est obtenue en exécutant le code DFO à la suite de la phase globale. Le cadre de la phase globale est donc rempli par l'Algorithme 4.4.

Enfin, la dernière méthode hybride possède une phase locale qui se compose de l'Algorithme 4.5. Cela revient à appliquer le code BCDFO une fois l'optimisation globale terminée.

En pratique, les tests et les choix de paramètres nous amèneront à considérer deux méthodes hybrides supplémentaires. Il s'agit de variantes des deux premiers algorithmes hybrides, qui s'obtiennent en jouant sur les points utilisés pour construire leur modèle RBF. Nous repréciserons les différents algorithmes testés dans le chapitre suivant à la Section 5.2.6.

Chapitre 5

Présentation des tests et résultats obtenus

Ce dernier chapitre regroupe ce qui se rapporte aux tests numériques effectués sur nos méthodes hybrides. Nous commençons par présenter les fonctions mathématiques que nous utilisons pour effectuer les tests. Ensuite, nous décrivons les différents choix de paramètres que nous avons réalisés. Finalement, nous présentons les résultats des tests numériques effectués sur un ensemble de fonctions mathématiques, ainsi que les profils de perfomance qui sont des outils utilisés pour comparer nos différents algorithmes.

5.1 Description des fonctions tests mathématiques

Les tests sur des fonctions mathématiques constituent une première étape dans l'évaluation des méthodes d'optimisation. Le but est de valider les méthodes avant d'aborder une phase de tests sur des cas, des problèmes réels. De tels tests pourront faire l'objet de travaux ultérieurs.

Pour évaluer les performances de nos méthodes hybrides, nous avons constitué un ensemble de fonctions tests. Certaines, comme Rosenbrock ou Rastrigin, sont des fonctions "classiques" souvent utilisées pour tester l'efficacité de méthodes d'optimisation. D'autres ont été choisies car elles apparaissaient dans des articles lus dans le cadre de ce travail ([28, 29, 30]). Enfin, d'autres fonctions proviennent de sites internet présentant des ensembles de fonctions tests ([40, 41, 42]).

L'ensemble de fonctions tests a été construit de telle sorte à contenir des fonctions de types différents. Certaines n'ont qu'un seul minimum (Rosenbrock par exemple), d'autres sont fortement multimodales (comme Rastrigin ou Ackley). Nous avons également des problèmes de petite dimension (2 à 6 variables), de dimension moyenne (10 variables) et de grande dimension (20 à 50 variables). Rappelons que nous nous trouvons dans le cadre de l'optimisation globale, d'où l'intérêt de considérer des fonctions multimodales.

Nous commençons par décrire chaque fonction individuellement, en donnant son expression, les bornes sur ses variables, la valeur de son optimum global ainsi qu'un graphique de la fonction et de ses courbes de niveau lorsque la fonction existe en dimension 2. Nous terminons cette section par un réca-

pitulatif des fonctions considérées pour les tests.

Ackley

La fonction Ackley est fortement multimodale. Son expression analytique est donnée par

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e,$$

où n représente le nombre de variables, $x \in \mathbb{R}^n$ et $e = \exp(1)$. Le domaine admissible est donné par

$$x_i \in [-5; 5] \quad i = 1, ..., n.$$

L'optimum global est situé en $x_i = 0$ pour $i = 1, ..., n$ et vaut 0. Une représentation de cette fonction en 2 dimensions, ainsi qu'un graphique de ses courbes de niveau, sont donnés à la Figure 5.1. Cette fonction a été optimisée lorsque le nombre de variables est de 2, 10 et 20.

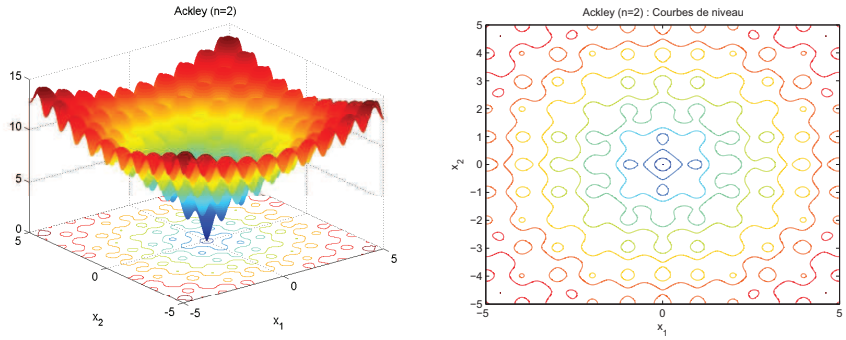


FIGURE 5.1 – Ackley (2 dimensions) : fonction et courbes de niveau

Branin

Branin est une fonction à 2 dimensions et qui possède 3 optima globaux. L'expression de la fonction est

$$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10,$$

où $x = (x_1,\ x_2) \in \mathbb{R}^2, x_1 \in [-5; 10]$ et $x_2 \in [0; 15]$. Les 3 minima (globaux) sont $x = (-\pi; 12.275)$, $x = (\pi; 2.275)$ et $x = (9.42477796; 2.47499998)$. La valeur de la fonction en ces points est 3.97887357729739e-1. La Figure 5.2 représente cette fonction ainsi que ses courbes de niveau.

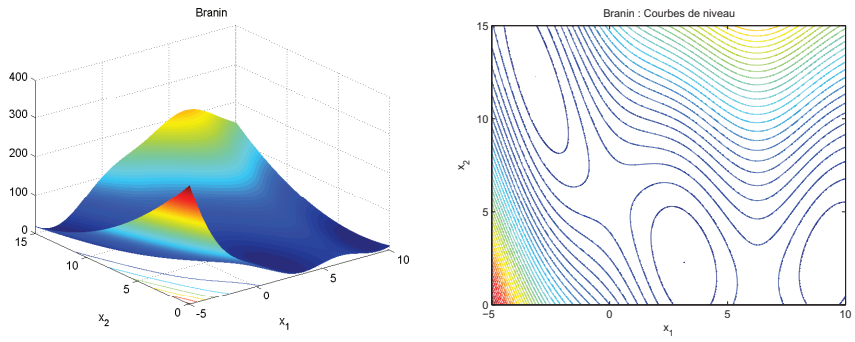


FIGURE 5.2 – Branin : fonction et courbes de niveau

Branin modifiée

La fonction de Branin modifiée comporte un terme supplémentaire par rapport à la fonction de Branin classique. Ce terme permet de ne plus avoir qu'un seul minimum global, les deux autres étant maintenant des minima locaux. L'expression analytique est donnée par

$$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10 + 5x_1,$$

où $x = (x_1,\ x_2) \in \mathbb{R}^2, x_1 \in [-5; 10]$ et $x_2 \in [0; 15]$. Le minimum global est situé en $x = (-\pi; 12.275)$ et vaut -16.64. Cette version modifiée de la fonction Branin est représentée à la Figure 5.3.

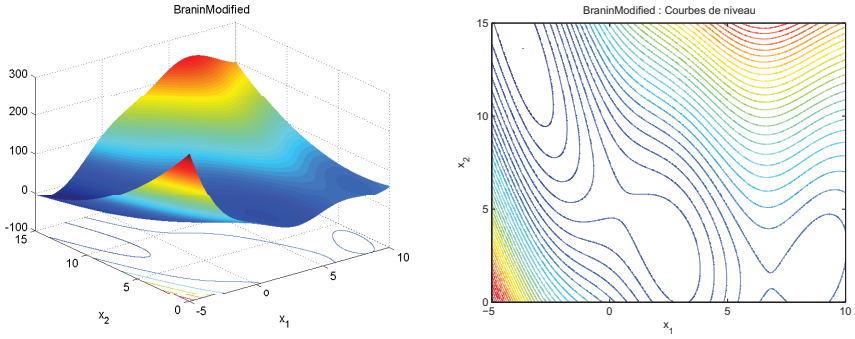


FIGURE 5.3 – Branin modifiée : fonction et courbes de niveau

Camelback

La fonction Camelback (6-humps) possède 2 variables. Elle est définie par

$$f(x) = x_1^2(4 - 2.1x_1^2 + \frac{x_1^4}{3}) + x_1x_2 + x_2^2(-4 + 4x_2^2),$$

où $x = (x_1,\ x_2) \in \mathbb{R}^2$. Le domaine admissible est délimité par

$$x_i \in [-2; 2] \quad i = 1, 2.$$

Camelback possède 6 minima dont 2 globaux : $x = (-0.08984201; 0.7126564)$ et $x = (0.08984201; -0.7126564)$. La valeur optimale de la fonction est -1.0316284535. La Figure 5.4 représente la fonction et ses courbes de niveau sur l'ensemble admissible. La Figure 5.5 constitue un "zoom" sur la vallée (i.e., approximativement $-1 \leq x_2 \leq 1$) afin de mieux représenter les optima.

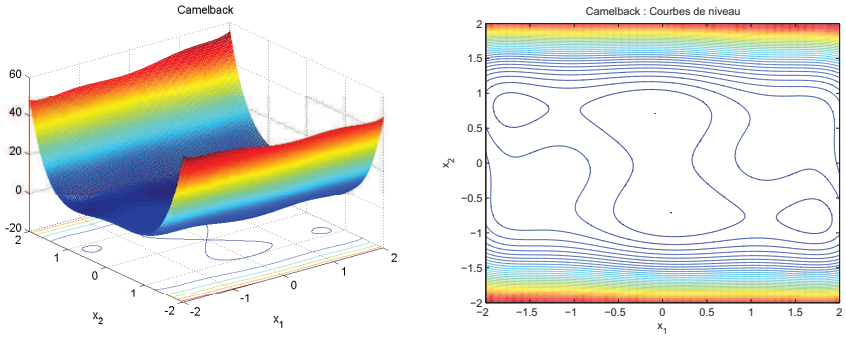


FIGURE 5.4 – Camelback : fonction et courbes de niveau

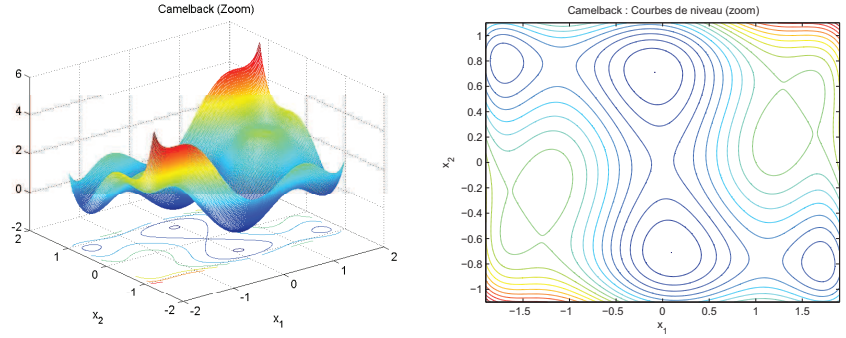


FIGURE 5.5 – Camelback : zoom sur la fonction et les courbes de niveau

Candle

L'expression de la fonction Candle est donnée par

$$f(x) = 0.5 - \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{1 + 0.001(x_1^2 + x_2^2)^2},$$

où $x = (x_1,\ x_2) \in \mathbb{R}^2$, $x_1 \in [-10; 10]$ et $x_2 \in [-10; 10]$. Elle possède une infinité d'optima mais un seul maximum global en $x = (0; 0)$, où la fonction vaut 1. Le graphique de la fonction est représenté à la Figure 5.6 tandis que les courbes de niveau, sur tout l'ensemble admissible puis autour de l'optimum global, sont illustrées à la Figure 5.7. Notons que cette fonction sera la seule de l'ensemble de fonctions tests à être *maximisée*.

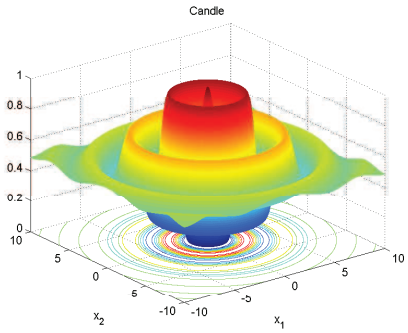


FIGURE 5.6 – Candle : fonction

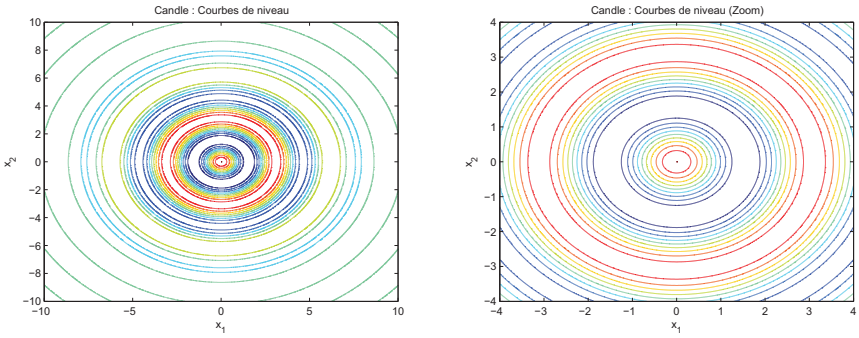


FIGURE 5.7 – Candle : courbes de niveau sur l'ensemble admissible et zoom autour du maximum global

Griewank

La fonction Griewank, très fortement multimodale, est définie par

$$f(x) = \sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1,$$

où n représente le nombre de variables et $x \in \mathbb{R}^n$. Le domaine admissible est donné par

$$x_i \in [-600; 600] \quad i = 1, ..., n.$$

L'optimum global est situé en $x_i = 0$ pour $i = 1, ..., n$ et vaut 0. La Figure 5.8 représente la fonction et ses courbes de niveaux sur tout l'espace de recherche tandis que la Figure 5.9 ne représente qu'une partie du domaine admissible afin de faire apparaître les nombreux optima de la fonction. Lors des tests, nous avons utilisé cette fonction en 2 et 20 dimensions.

Hartman

Hartman est une fonction de dimension n, définie par

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{n} A_{ij}(x_j - P_{ij})^2\right),$$

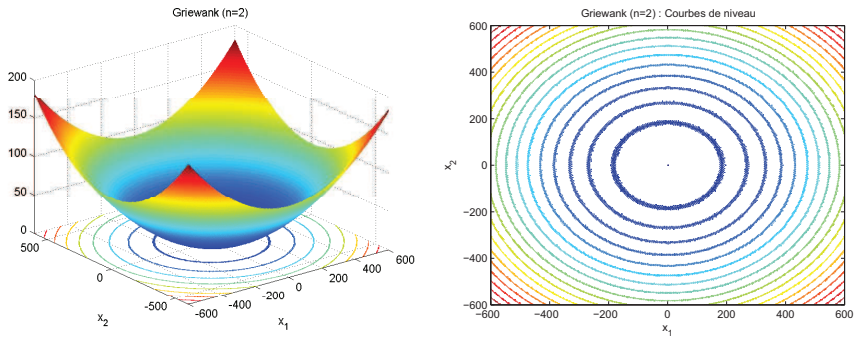


FIGURE 5.8 – Griewank (2 dimensions) : fonction et courbes de niveau

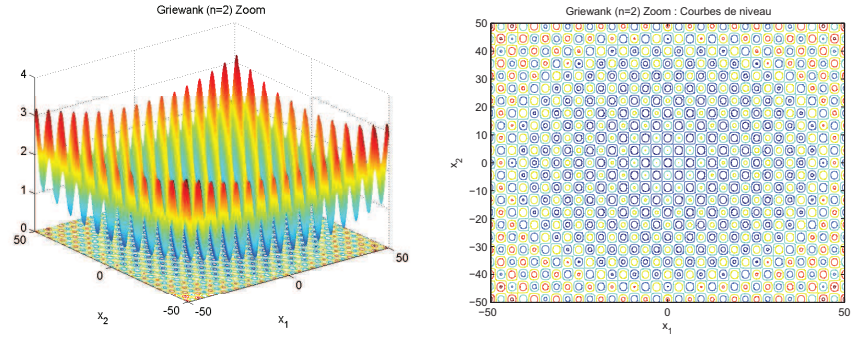


FIGURE 5.9 – Griewank (2 dimensions) : zoom sur la fonction et les courbes de niveau

où $x \in \mathbb{R}^n$, $c = (1,\ 1.2,\ 3,\ 3.2)$, $A \in \mathbb{R}^{4\times n}$ et $P \in \mathbb{R}^{4\times n}$. Le domaine de recherche est défini par

$$x_i \in [0;1] \quad i = 1, ..., n.$$

La fonction Hartman a été testée pour deux valeurs de n différentes : 3 et 6.

En dimension 3, les matrices A et P valent

$$A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix} \quad \text{et} \quad P = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix}.$$

La fonction possède 4 minima locaux et un global. Il est situé en $x = (0.114614; 0.555649; 0.852547)$ et a -3.86278 comme valeur objectif.

En dimension 6, la matrice A vaut

$$A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$$

et la matrice P est donnée par

$$P = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}.$$

Hartman6 présente 6 minima locaux. Le minimum global est atteint au point $x = (0.20169; 0.150011; 0.476874; 0.275332; 0.311652; 0.6573)$ où la fonction vaut -3.32237.

Hosaki

Hosaki est une fonction bi-dimensionnelle définie par

$$f(x) = \left(1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4\right) x_2^2 e^{-x_2},$$

où $x = (x_1,\ x_2) \in \mathbb{R}^2$ et $x_i \in [0; 5]$ pour $i = 1, 2$. Cette fonction possède un minimum local en $x = (1; 2)$ et un minimum global en $x = (4; 2)$ où la fonction vaut -2.3458115761. La Figure 5.10 représente cette fonction et ses courbes de niveaux.

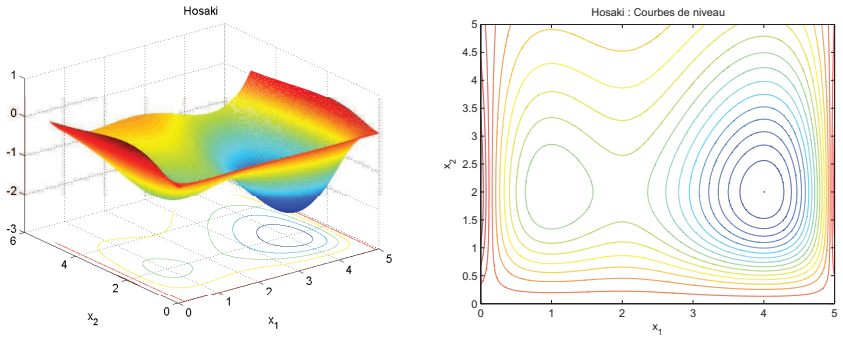


FIGURE 5.10 – Hosaki : fonction et courbes de niveau

Mystery

La fonction Mystery possède deux variables et son expression est donnée par

$$f(x) = 2 + 0.01(x_2 - x_1^2)^2 + (1 - x_1)^2 + 2(2 - x_2)^2 + 7\sin(0.5x_1)\sin(0.7x_1x_2),$$

où $x = (x_1, x_2) \in \mathbb{R}^2$ et $x_i \in [-0.5; 5]$ pour $i = 1, 2$. La fonction présente 2 optima locaux et un minimum global, en $x = (2.504425207128; 2.577837766187)$. Mystery en ce point vaut -1.456525819489. La Figure 5.11 représente les graphiques de la fonction et de ses courbes de niveau.

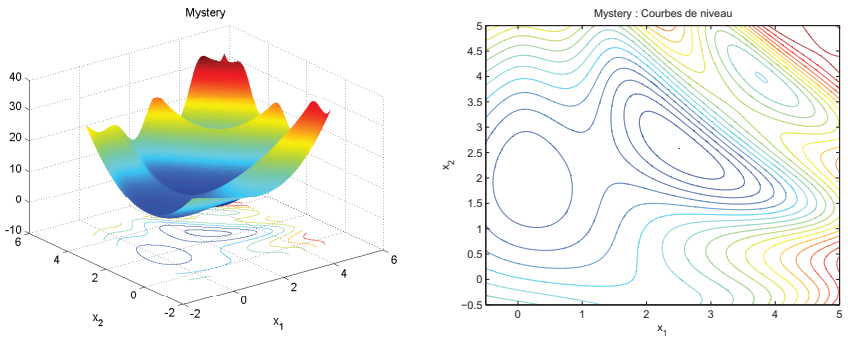


FIGURE 5.11 – Mystery : fonction et courbes de niveau

Rastrigin

La fonction de Rastrigin est multidimensionnelle. Son expression analytique est

$$f(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)) + 10n,$$

où n représente la dimension de la fonction, $x \in \mathbb{R}^n$. L'espace de recherche est défini par

$$x_i \in [-5; 5] \quad i = 1, ..., n.$$

Il s'agit d'une fonction possédant de nombreux optima locaux. Son minimum global se situe en $x_i = 0$ pour $i = 1, ..., n$ et la fonction y est nulle. Dans les tests, la dimension a été fixée à 2 et à 20. La fonction bi-dimensionnelle ainsi que ses courbes de niveaux sont représentées à la Figure 5.12. La Figure 5.13 représente quant à elle un zoom de la région du minimum global.

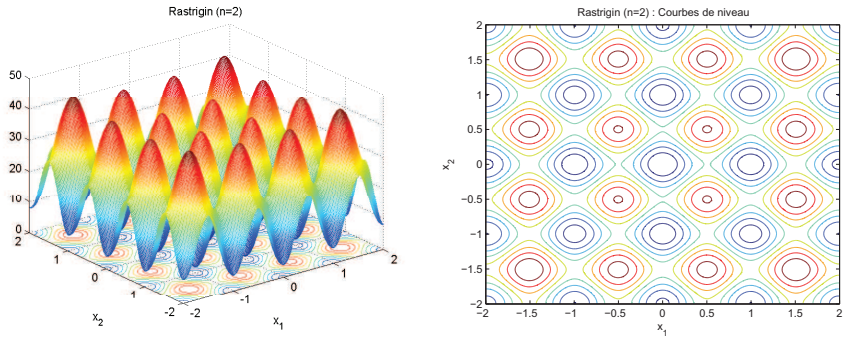


FIGURE 5.12 – Rastrigin (2 dimensions) : fonction et courbes de niveau

Rosenbrock

L'expression de la fonction de Rosenbrock à n dimensions est donnée par

$$f(x) = \sum_{i=1}^{n-1} \left((1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2 \right),$$

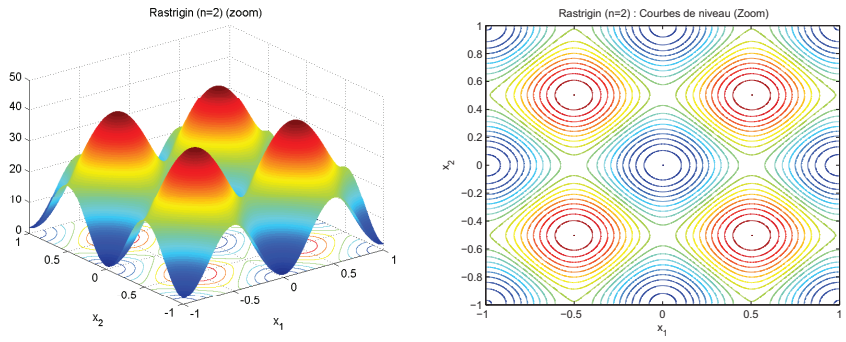


FIGURE 5.13 – Rastrigin (2 dimensions) : zoom sur la fonction et les courbes de niveau autour de l'optimum global

où n représente la dimension, $x \in \mathbb{R}^n$ et $x_i \in [-2; 2]$ pour $i = 1, ..., n$. Le minimum global de cette fonction est situé au point $x_i = 1$ pour $i = 1, ..., n$. La fonction y vaut zéro. La Figure 5.14 réprésente cette fonction en dimension 2. Dans les tests, la dimension 2, mais aussi les dimensions 30 et 50, ont été utilisées.

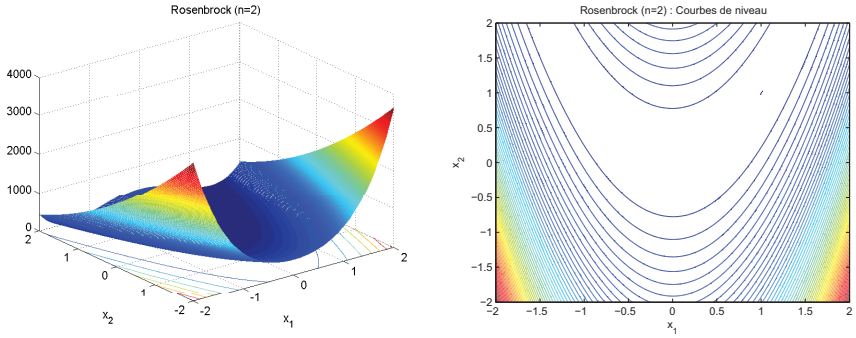


FIGURE 5.14 – Rosenbrock (2 dimensions) : fonction et courbes de niveau

Schwefel7

La fonction Schwefel7 est fortement multimodale. Son expression, en fonction du nombre de variables n, est donnée par

$$f(x) = \sum_{i=1}^{n} \left(-x_i \sin(\sqrt{|x_i|}) \right),$$

où $x \in \mathbb{R}^n$ et $x_i \in [-500; 500]$ pour $i = 1, ..., n$. Le minimum global est atteint en $x_i = 420.9687$ pour $i = 1, ..., n$ et dépend de la dimension de la fonction. Il vaut -418.9829 n. Pour les tests, la dimension a été fixée à $n = 10$. La représentation de cette fonction en dimension 2 est illustrée à la Figure 5.15.

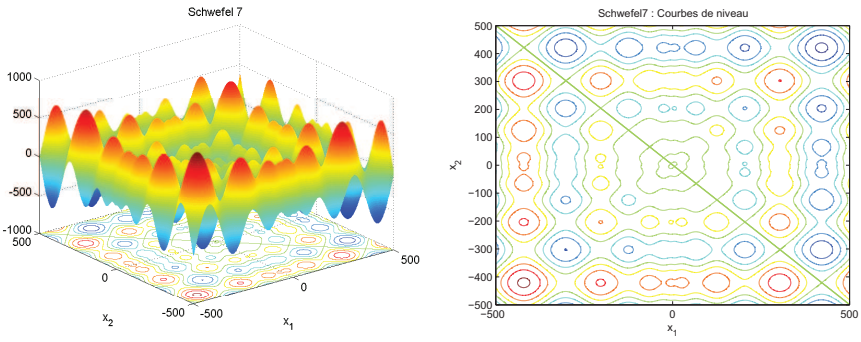


FIGURE 5.15 – Schwefel7 (2 dimensions) : fonction et courbes de niveau

Sphere

La fonction Sphere est donnée par

$$f(x) = \sum_{i=1}^{n} x_i^2,$$

où n correspond au nombre de variables du problème, $x \in \mathbb{R}^n$ et $x_i \in [-5; 5]$ pour $i = 1, ..., n$. Cette fonction possède un seul minimum au point $x_i = 0$ pour $i = 1, ..., n$. La fonction y est nulle. Une représentation en deux dimensions est reprise à la Figure 5.16. Lors des tests, la fonction a été minimisée pour deux valeurs de n : 2 et 10.

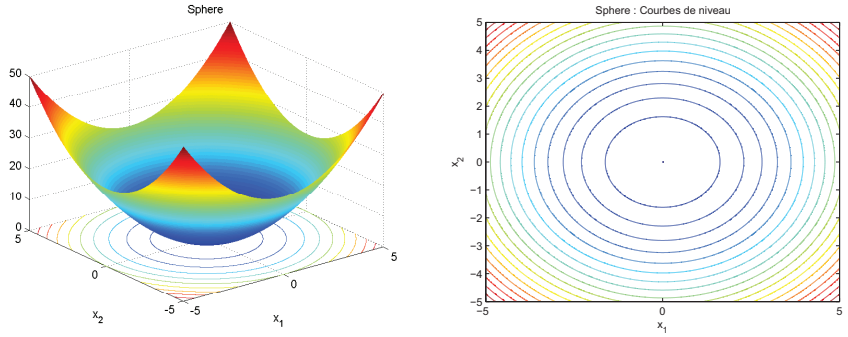


FIGURE 5.16 – Sphere (2 dimensions) : fonction et courbes de niveau

Tableau récapitulatif

Le Tableau 5.1 reprend les 21 fonctions que nous avons employées pour les tests numériques. Pour chacune, le nom, le nombre de variables et le type de fonction sont indiqués. Par type de fonction, nous entendons unimodale (un seul optimum), multimodale[i] (quelques optima dont i sont globaux) ou fortement multimodale (de nombreux optima dont un seul est global).

Nom	Dimension	Type
Ackley2	2	fortement multimodale
Ackley10	10	fortement multimodale
Ackley20	20	fortement multimodale
Branin	2	multimodale[3]
Branin modifiée	2	multimodale[1]
Camelback	2	multimodale[2]
Candle	2	fortement multimodale
Griewank2	2	fortement multimodale
Griewank20	20	fortement multimodale
Hartman3	3	multimodale[1]
Hartman6	6	multimodale[1]
Hosaki	2	multimodale[1]
Mystery	2	multimodale[1]
Rastrigin2	2	fortement multimodale
Rastrigin20	20	fortement multimodale
Rosenbrock2D	2	unimodale
Rosenbrock30	30	unimodale
Rosenbrock50	50	unimodale
Schwefel710	10	fortement multimodale
Sphere2	2	unimodale
Sphere10	10	unimodale

TABLE 5.1 – Tableau récapitulatif des fonctions tests.

5.2 Choix des paramètres

Les paramètres qui interviennent dans nos méthodes hybrides sont nombreux. Le but de cette section est de préciser les différentes valeurs que nous avons choisies pour réaliser nos tests. Certains paramètres ont été fixés de façon identique pour tous les problèmes, d'autres prennent des valeurs différentes suivant les fonctions. La description des valeurs des paramètres est d'abord détaillée pour le plan d'expériences, ensuite pour l'algorithme génétique et enfin pour les différentes méthodes locales.

5.2.1 Le DoE

Deux choix peuvent influencer le DoE à partir duquel nous travaillons. Le premier est la méthode appliquée afin de générer les points. Nous utilisons la méthode LCVT puisqu'il s'agit de celle qui permet le meilleur recouvrement de l'espace de recherche (cfr. Chapitre 2). Cette même méthode sera appliquée pour tous les problèmes étudiés.

Le deuxième facteur dont dépend le DoE est le nombre de points dont il est composé. Ce nombre de points ne sera pas identique pour tous les problèmes et sera notamment influencé par le nombre de variables de la fonction. La pratique la plus courante à Cenaero est de choisir un nombre de points pour constituer le DoE situé entre 2 et 5 fois le nombre de variables du

problème (car, en général, la fonction objectif est très coûteuse). On obtient ainsi les valeurs reprises dans le Tableau 5.2. On remarquera que les valeurs pour les fonctions Ackley2, Candle et Rastrigin2 sont un peu plus élevées mais ce choix est justifié par la forte multimodalité de ces fonctions.

Fonction	DoE	Fonction	DoE	Fonction	DoE
Ackley2	20	Griewank2	10	Rastrigin20	40
Ackley10	30	Griewank20	40	Rosenbrock2D	10
Ackley20	40	Hartman3	15	Rosenbrock30	60
Branin	10	Hartman6	20	Rosenbrock50	100
Branin modifiée	10	Hosaki	6	Schwefel710	30
Camelback	6	Mystery	6	Sphere2	10
Candle	20	Rastrigin2	20	Sphere10	30

TABLE 5.2 – Tableau du nombre de points constituant le DoE pour chaque fonction test.

5.2.2 L'algorithme génétique

L'algorithme génétique était déjà implémenté dans Minamo. Une valeur par défaut de ses paramètres était donc disponible. Pour la plupart des paramètres, nous avons conservé ces valeurs par défaut. C'est le cas, par exemple, du taux de mutation ou du nombre d'individus conservés par l'élitisme.

Trois paramètres ont été choisis de façon dépendante à la dimension du problème optimisé : la taille de la population, le nombre de générations et le nombre maximum d'itérations de la boucle externe de l'algorithme génétique (cfr. Figure 3.3). Ces valeurs de paramètres seront plus élevées pour des problèmes dont la dimension est plus grande.

La taille de la population a été fixée à 50 pour les problèmes de dimension 2 à 10 et à 100 pour les fonctions qui possèdent plus de 10 variables. Le nombre de générations est fixé selon le même critère : 50 générations pour les problèmes de dimension inférieure à 10 (inclus) et 100 générations pour les autres.

Le choix du nombre d'itérations est plus délicat. Il est directement lié au nombre d'évaluations de la fonction objectif que l'on attribue à la méthode globale puisqu'une évaluation est réalisée au cours de chaque itération. Ce nombre dépendra donc de la dimension du problème mais surtout de son type (i.e., problème plus ou moins multimodal). Nous avons choisi de fixer le nombre d'itérations de telle sorte que les évaluations de fonction utilisées au cours de la construction du DoE et de l'exécution de l'algorithme génétique représentent la moitié du nombre maximum d'évaluations autorisées. Ce choix est basé sur les tests réalisés dans les articles [28, 29, 30]. Le Tableau 5.3 regroupe les valeurs de paramètres fixées pour chaque fonction, où Pop. désigne la taille de la population, Gén. le nombre de générations et It. le nombre d'itérations.

Fonction	Pop.	Gén.	It.	Fonction	Pop.	Gén.	It.
Ackley2	50	50	30	Hosaki	50	50	14
Ackley10	50	50	40	Mystery	50	50	14
Ackley20	200	100	60	Rastrigin2	50	50	30
Branin	50	50	20	Rastrigin20	200	100	60
Branin modifiée	50	50	20	Rosenbrock2D	50	50	30
Camelback	50	50	14	Rosenbrock30	200	100	60
Candle	50	50	30	Rosenbrock50	200	100	60
Griewank2	50	50	20	Schwefel710	50	50	40
Griewank20	200	100	60	Sphere2	50	50	20
Hartman3	50	50	25	Sphere10	50	50	40
Hartman6	50	50	30				

TABLE 5.3 – Tableau des paramètres de l'algorithme génétique pour chaque fonction test.

5.2.3 Nos méthodes de région de confiance

Comme nous avons implémenté deux méthodes de région de confiance nous-mêmes, spécifier qu'on utilise des paramètres par défaut n'est pas suffisant, il faut également les déterminer. Les deux méthodes ayant des implémentations différentes, elles n'auront pas tous les mêmes paramètres. Nous commençons par traiter ceux qui sont communs aux deux et ensuite ceux spécifiques à chaque méthode. Pour plus de simplicité dans la suite de ce travail, nous noterons TR la première méthode locale qui utilise un modèle quadratique (cfr. Section 4.1.5) et TR2 la seconde méthode de région de confiance qui utilise un modèle RBF (cfr. Section 4.1.6).

Paramètres communs aux deux méthodes

Le rayon initial de la région de confiance a été fixé par défaut à la valeur 1. C'est cette valeur qui sera utilisée lors des tests, sauf pour les fonctions Hartman3 et Hartman6. En effet, les variables de ces fonctions sont comprises dans l'intervalle $[0; 1]$. Un rayon de la région de confiance de longueur 1 a donc peu de sens puisqu'il donne lieu à une région de confiance initiale qui sort de l'espace de recherche. Pour ces deux fonctions, le rayon initial sera donc différent de la valeur par défaut et fixé à 0.1.

Nous avons choisi de placer une borne supérieure sur la taille du rayon de la région de confiance. Pour tous les tests, nous utilisons la valeur par défaut qui est de 1000. Celle-ci peut paraître élevée mais rappelons que certains problèmes peuvent avoir un espace de recherche relativement grand, comme Schwefel710 par exemple.

Les paramètres η_1 et η_2 de l'Algorithme 4.1 doivent également être fixés. Suivant la méthode retenue pour mettre à jour le rayon de la région de confiance (classique ou en fonction de la norme du pas), les paramètres α_1 et α_2 présentés à la Section 4.1.4 peuvent également intervenir. Afin de déterminer un jeu de valeurs fixées par défaut, nous avons testé trois configurations

de paramètres sur la fonction Rosenbrock. Les trois cas possibles sont précisés ci-dessous.

	Cas 1	Cas 2	Cas 3
Rayon	classique	pas	pas
η_1	0.01	0.01	0.0001
η_2	0.9	0.9	0.99
α_1	—	2.5	3.5
α_2	—	0.25	0.25

La configuration qui a donné les meilleurs résultats et qui a été retenue comme jeu de paramètres par défaut est la seconde. Il s'agit des valeurs de paramètres suggérées dans [5]. Il s'agit du choix que nous avons effectué dans un premier temps. Nous expliquerons dans la suite de cette section la modification que nous y avons apportée ainsi que la raison de ce changement.

Une autre option commune aux deux méthodes et laissée au choix de l'utilisateur est la façon de construire le modèle RBF (qui servira de fonction objectif dans la méthode TR et de modèle dans la méthode TR2). La première possibilité est de construire le modèle avec tous les points de la base de donnée. L'autre méthode consiste à ne sélectionner que $n+1$ points au minimum, situés dans un voisinage plus ou moins élargi [1] de la région de confiance, n étant le nombre de variables du problème étudié. Dans les deux cas, la base de données est filtrée au préalable afin d'éviter d'obtenir deux points exactement les mêmes dans l'ensemble qui sert à construire le modèle. Un paramètre caractérise l'écart minimum requis entre deux points. Il a été fixé à 10^{-14}.

Fonction	$\#f$	Fonction	$\#f$	Fonction	$\#f$
Ackley2	50	Griewank2	30	Rastrigin20	100
Ackley10	70	Griewank20	100	Rosenbrock2D	40
Ackley20	100	Hartman3	40	Rosenbrock30	120
Branin	30	Hartman6	50	Rosenbrock50	100
Branin modifiée	30	Hosaki	20	Schwefel710	70
Camelback	20	Mystery	20	Sphere2	30
Candle	50	Rastrigin2	50	Sphere10	70

TABLE 5.4 – Nombre d'évaluations de la fonction objectif exacte attribué à chaque fonction pour la méthode locale.

Finalement, le dernier point commun des deux méthodes au niveau des paramètres est le nombre maximum d'itérations réalisées (itérations externes dans le cas de TR). Celui-ci correspond au nombre d'évaluations de la fonction objectif puisqu'une seule évaluation a lieu par itération. Comme nous l'avons précisé à la Section 5.2.2, la répartition du nombre d'évaluations de fonction entre méthode globale (DoE compris) et locale est de moitié pour l'une, moitié pour l'autre. Au vu des valeurs choisies pour les DoE au Tableau 5.2 et pour l'algorithme génétique au Tableau 5.3, on obtient les valeurs reprises dans le Tableau 5.4 pour les méthodes de région de confiance.

1. Le voisinage sera élargi jusqu'à contenir le minimum de $n+1$ points.

Paramètres spécifiques à TR

La méthode choisie par défaut pour calculer le pas dans la méthode TR est l'algorithme de Steihaug-Toint. C'est la première méthode que nous avons utilisée pour résoudre ce problème. Par la suite, nous avons testé l'utilisation d'IPOPT pour minimiser le modèle quadratique dans la région de confiance. Cependant, cela est beaucoup plus coûteux en temps que la méthode de Steihaug-Toint, sans apporter beaucoup plus au niveau de la qualité de la solution. C'est pourquoi nous utilisons la méthode de Steihaug-Toint.

Un second paramètre spécifique à la méthode TR est le nombre maximum d'itérations internes qui sont effectuées (sur un même RBF). Après avoir réalisé différents tests, nous avons constaté que peu d'itérations étaient nécessaires pour obtenir la convergence vers un optimum du modèle RBF. C'est pourquoi le nombre maximal d'itérations internes a été fixé à 20.

Enfin, les derniers paramètres qui ont été fixés pour la méthode TR sont les précisions utilisées dans les critères d'arrêt sur le gradient et le pas relatifs (cfr. Section 4.1.5). Notre choix est basé sur les valeurs par défaut suggérées dans [7]. La valeur proposée pour le paramètre *gradtol* est la racine cubique de la précision machine, i.e., 6e-6 dans notre cas. Nous avons donc choisi la valeur 10^{-6}. Pour *steptol*, les auteurs proposent $eps^{2/3}$ où *eps* représente la précision machine, ce qui équivaut à 3.66e-11. Nous avons donc fixé ce paramètre à 10^{-11}.

Paramètres spécifiques à TR2

Les paramètres spécifiques à la méthode TR2 viennent de l'algorithme utilisé afin de résoudre le sous-problème de la région de confiance : IPOPT. La plupart des options d'IPOPT ont conservé leurs valeurs par défaut. IPOPT nécessite un solveur local. Nous avons utilisé le solveur de base, MA27. Le paramètre pour la convergence a été légèrement assoupli et fixé à 10^{-6}, au lieu de 10^{-8} par défaut. Un élément important est le nombre maximal d'itérations laissé à l'algorithme. Au plus celui-ci est élevé, au plus les résultats obtenus seront meilleurs. Cependant, cela peut vite devenir coûteux en temps d'exécution. Après différents tests, nous avons donc choisi de fixer le nombre maximal d'itérations à 200.

Suite à l'observation des résultats des premiers tests, nous avons parfois constaté un comportement étrange de la méthode TR2. Nous avons alors réalisé que l'algorithme IPOPT ne parvenait pas toujours à déterminer une solution correcte au sous-problème de la région de confiance dans le nombre d'itérations imparti. Le pas défini par ce nouvel itéré n'avait donc pas de sens et le fait de le prendre en compte dans la mise à jour du rayon de la région de confiance impliquait des erreurs de méthodes. C'est pourquoi nous avons modifié la méthode de mise à jour par défaut de l'algorithme TR2 et l'avons dorénavant fixée à la procédure classique (cfr. Section 4.1.4).

5.2.4 L'algorithme DFO

La plupart des paramètres de l'algorithme DFO ont été fixés de façon identique pour tous les problèmes. On obtient ainsi les valeurs qui sont re-

prises dans le Tableau 5.5. Signalons une exception au niveau de la valeur initiale du rayon pour les fonctions Hartman3 et Hartman6, comme à la Section 5.2.3.

Paramètre	Valeur
Nombre de points initiaux	tous les points disponibles
Valeur de f au(x) point(s) initial(aux)	.TRUE.
Nombre max d'itérations	1000
Critère d'arrêt	1
Rayon minimum	0.00001
Tolérance de faisabilité	0.00001
Rayon initial	1
Scaling	aucun

TABLE 5.5 – Paramètres de l'algorithme DFO.

Le premier paramètre indique que tous les points évalués avant le début de DFO (provenant du DoE et de la recherche locale) sont fournis à l'algorithme, avec la valeur de fonction qui leur correspond (grâce à la valeur du second paramètre). Le critère d'arrêt égal à 1 équivaut à stopper l'algorithme lorsque le rayon de la région de confiance devient inférieur à la valeur minimale donnée. Finalement, la tolérance de faisabilité est un paramètre utilisé lors de la résolution du sous-problème contraint de la région de confiance.

Il reste un dernier paramètre de l'algorithme DFO dont nous n'avons pas encore parlé : le nombre maximal d'évaluations de fonction. Comme pour la section précédente, celui-ci correspond au nombre d'évaluations attribuées à la méthode locale, qui diffèrent d'un problème à l'autre (cfr. Tableau 5.4). En pratique, ce nombre étant beaucoup plus faible que le nombre maximal d'itérations, c'est ce dernier qui sera responsable d'un arrêt de l'algorithme lorsqu'il n'y a pas convergence.

5.2.5 L'algorithme BCDFO

La façon de fixer les paramètres de l'algorithme BCDFO est identique à celle utilisée pour l'algorithme DFO [2]. Rappelons que le code BCDFO est normalement utilisé avec un seul point initial. Or, nous voudrions passer à l'algorithme tous les points qui ont été évalués auparavant, comme avec l'algorithme DFO. Afin de pouvoir transmettre à l'algorithme toute l'information sur les points évalués par la fonction objectif durant les étapes précédentes de la méthode hybride, nous avons utilisé l'option *restart* de BCDFO. Celle-ci est utilisée en temps normal pour relancer un algorithme qu'on a interrompu avec les paramètres tels qu'ils étaient au moment de l'interruption. Pour atteindre notre objectif, il suffit de fixer les paramètres à leur valeur initiale et de donner non pas un point initial mais l'ensemble de points contenus dans la base de données. Les valeurs des différents paramètres sont données dans le Tableau 5.6. BCDFO possède de nombreux

2. La remarque sur le rayon initial des fonctions Hartman3 et Hartman6 est toujours valable.

paramètres, seuls les plus importants sont indiqués. Pour les autres, les valeurs par défaut ont été utilisées.

Paramètre	Valeur
Rayon Initial	1
Nombre max d'itérations	1000
Solveur local	Gradient-Conjugué
Précision critère d'arrêt	1e-5
Degré m_0	linéaire
Degré m_R	linéaire
Restart	activé

TABLE 5.6 – Paramètres de l'algorithme BCDFO.

La précision du critère d'arrêt correspond à la précision à atteindre par la norme du gradient du modèle et l'erreur sur le modèle pour pouvoir déclarer la convergence. Les deux paramètres suivants représentent respectivement le nombre minimum de points qui constituent l'ensemble d'interpolation initial puis dans les itérations utlérieures. Un degré linéaire correspond à un modèle construit à partir de $n + 1$ points.

Finalement, un dernier paramètre important est le nombre maximal d'évaluations de fonction autorisé pour l'algorithme BCDFO. Ce nombre sera identique à celui des méthodes locales précédentes, afin d'obtenir des résultats comparables. Ces valeurs sont reprises au Tableau 5.4.

5.2.6 L'algorithme hybride

Le principal paramètre de l'algorithme hybride correspond à la répartition des évaluations de la fonction objectif entre les méthodes globale et locale. Nous avons spécifié aux sections précédentes qu'une moitié des évaluations était consacrée à la méthode globale (y compris le DoE) tandis que l'autre moitié était réservée pour la méthode locale. Le Tableau 5.7 synthétise ces différentes valeurs et indique également le nombre total d'évaluations de fonction autorisées pour chacun des problèmes tests.

Au vu des différentes méthodes et paramètres disponibles, nous allons exécuter et comparer 8 algorithmes : 2 algorithmes génétiques et 6 méthodes hybrides. Le premier algorithme génétique sera noté PureGA car il s'agit d'un algorithme génétique (assisté par modèles) "classique". Le second algorithme génétique testé fait intervenir l'option de Move-Limit implémentée dans Minamo et que nous avons décrite à la Section 2.2.3. Cette option "imite" le comportement d'une méthode locale en limitant les points utilisés pour la construction du modèle RBF optimisé ensuite par l'algorithme génétique. Pour obtenir quelque chose de comparable avec nos méthodes hybrides, l'option de Move-Limit ne sera activée qu'à partir de l'itération qui correspond au début de la méthode locale pour les méthodes hybrides. Cette méthode sera notée MLGA.

Nous disposons de 4 méthodes locales. Pour obtenir 6 hybrides nous allons considérer les méthodes de région de confiance, que nous avons implémentées,

Nom	DoE	Génétique	Local	Total
Ackley2	20	30	50	100
Ackley10	30	40	70	140
Ackley20	40	60	100	200
Branin	10	20	30	60
Branin modifiée	10	20	30	60
Camelback	6	14	20	40
Candle	20	30	50	100
Griewank2	10	20	30	60
Griewank20	40	60	100	200
Hartman3	15	25	40	80
Hartman6	20	30	50	100
Hosaki	6	14	20	40
Mystery	6	14	20	40
Rastrigin2	20	30	50	100
Rastrigin20	40	60	100	200
Rosenbrock2D	10	30	40	80
Rosenbrock30	60	60	120	240
Rosenbrock50	100	60	100	260
Schwefel710	30	40	70	140
Sphere2	10	20	30	60
Sphere10	30	40	70	140

TABLE 5.7 – Nombre d'évaluations de la fonction objectif attribué à chaque partie de l'algorithme hybride, ainsi que le nombre total.

une fois avec un modèle construit en utilisant tous les points puis une seconde fois en limitant les points utilisés dans un voisinage de la région de confiance. En appliquant ces méthodes après un algorithme génétique, on obtient ainsi les méthodes GATR Full et GATR2 Full dans le premier cas et les méthodes GATR Local et GATR2 Local dans le second. La méthode qui combine un algorithme génétique et l'algorithme DFO est appelée GADFO. Enfin, l'application de l'algorithme BCDFO après l'algorithme génétique donne lieu à la méthode notée GABCDFO. Précisons que dans le cas de toutes les méthodes hybrides, l'algorithme génétique qui est employé ne contient pas de Move-Limit. Le but du génétique étant uniquement de remplir un rôle d'exploration de l'espace de recherche.

5.3 Présentation des profils de performance

Les *profils de performance* sont un outil pour comparer différents algorithmes. Il s'agit d'une représentation graphique d'une fonction de distribution. Le calcul de celle-ci se base sur un critère, le temps CPU d'exécution de l'algorithme ou le nombre d'évaluations de la fonction objectif utilisés, par exemple. Leur principe est présenté dans [20].

On suppose que l'on veut comparer n_a algorithmes sur un ensemble $\mathcal{P}$ de

$|\mathcal{P}|$ problèmes tests. Pour chaque problème et chaque algorithme, on réalise une mesure de performance. La mesure pour le problème p avec l'algorithme a est notée $t_{p,a}$. On calcule ensuite, pour chaque problème, la mesure minimale

$$best_p = \min_a \{t_{p,a}\}. \tag{5.1}$$

On peut maintenant calculer le ratio de performance. Pour chaque algorithme et chaque problème, la mesure de performance est divisée par la meilleure valeur associée au problème correspondant. Le ratio est défini par

$$r_{p,a} = \frac{t_{p,a}}{best_p}.$$

Si un algorithme ne permet pas de résoudre un problème, la mesure de performance sera infinie. La valeur du ratio est alors posée égale à une constante r_{max} qui est supérieure ou égale à toutes les valeurs des autres ratios.

Les fonctions qui apparaissent sur le graphique sont définies par

$$\rho_a(\tau) = \frac{1}{|\mathcal{P}|} \#\{p \in \mathcal{P} \mid r_{p,a} \leq \tau\}, \tag{5.2}$$

pour chaque algorithme a étudié. La notation $\#$ représente le nombre d'éléments contenus dans l'ensemble. La fonction $\rho(\tau)$ correspond à la fonction de distribution cumulative (en escalier) du ratio de performance.

Deux caractéristiques importantes d'un algorithme lorsqu'on l'étudie sont sa robustesse et son efficacité. L'efficacité correspond au nombre de problèmes pour lesquels l'algorithme est le meilleur. Cette mesure correspond à $\rho(1)$, par construction de la fonction. Si on travaille en échelle logarithmique, l'efficacité sera l'ordonnée du point d'abscisse $\log(1) = 0$. La robustesse d'un algorithme est sa capacité à résoudre un grand nombre de problèmes (variés), quelle que soit la valeur de performance. C'est le fait d'obtenir une solution qui est pris en compte. La robustesse peut aussi se lire sur le profil, pour les abscisses les plus grandes.

En général, au plus une courbe croît rapidement sur le profil, au plus l'algorithme est performant. Si un algorithme résout la totalité des problèmes tests, la courbe atteindra la valeur 1. Sinon, la plus haute valeur atteinte par la courbe indiquera la proportion de problèmes résolvables par l'algorithme.

La construction des profils de ce travail a été réalisée au moyen du script de Moré, disponible sur internet (cfr [39]). Comme mesure de performance $t_{p,a}$, nous avons choisi le nombre d'évaluations de fonction nécessaire pour vérifier le critère

$$f(x_0) - f(x) \geq (1 - tol)(f(x_0) - f_L), \tag{5.3}$$

où $f(x_0)$ représente la valeur de la fonction objectif au point initial, f_L est calculée pour chaque problème de $\mathcal{P}$ comme la meilleure valeur de f obtenue parmi les algorithmes testés (dans le nombre maximal d'évaluations de fonctions qui a été fixé), et tol correspond à la précision souhaitée.

5.4 Présentation des résultats

Les résultats que nous présentons dans cette section ont été obtenus en lançant 50 runs des algorithmes décrits à la Section 5.2.6 sur l'ensemble des 21 fonctions tests présentées à la Section 5.1. Pour la fonction Rosenbrock généralisée à 50 dimensions, nous n'avons réalisé que 30 runs pour chacune des différentes méthodes. En effet, l'optimisation de cette fonction dure assez longtemps à cause de son grand nombre de variables et 30 runs permettent quand même d'obtenir des résultats statistiques sensés. [3]

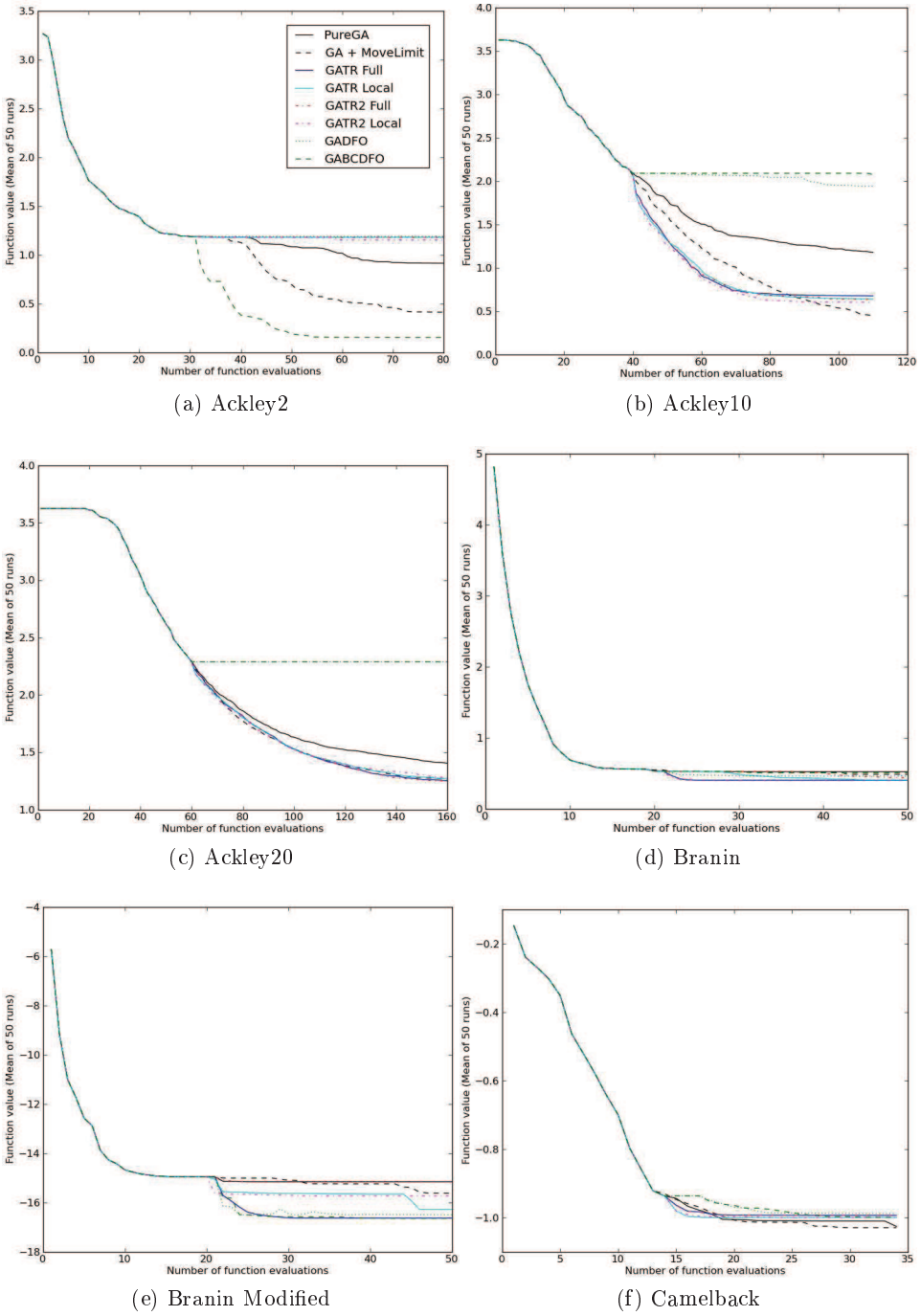
Pour réaliser une première analyse des résultats, nous avons observé le comportement des algorithmes pour chaque problème séparément. Pour chaque fonction, nous avons tracé un graphique qui représente la valeur moyenne de la fonction objectif (sur les différents runs) en fonction du nombre d'évaluations de cette fonction. Nous avons ainsi obtenu les graphiques représentés aux Figures 5.17, 5.18, 5.19 et 5.20. Pour les fonctions Rosenbrock2, Sphere2 et Sphere10, les valeurs de fonctions sont représentées suivant une échelle logarithmique afin de pouvoir distinguer les courbes des différents algorithmes.

Nous remarquons que le comportement des différentes méthodes varie suivant le type de la fonction considérée. Pour des problèmes simples qui possèdent peu de variables, tels que Branin, Camelback, Hartman3, Hartman6 et Hosaki, les courbes des méthodes sont très proches les unes des autres. D'autres problèmes de petites dimensions présentent des comportements plus distincts pour des méthodes différentes. C'est le cas par exemple de la version modifiée de la fonction de Branin, où les algorithmes hybrides utilisant DFO, BCDFO et TR Full donnent de meilleurs résultats en moyenne. Pour la fonction Mystery, c'est la méthode où le code DFO est appliqué qui se révèle la plus performante.

Dans l'ensemble, les méthodes GADFO et GABCDFO obtiennent de bons résultats pour les problèmes de petite dimension tandis que les méthodes qui utilisent des modèles RBF et que nous avons implémentées se retrouvent moins bien classées. C'est le cas par exemple pour la fonction de Rosenbrock à 2 variables. Cette moins bonne performance pourrait s'expliquer par une difficulté de représenter la vallée très plate de cette fonction dans la région du minimum par un RBF. Nos implémentations ne possèdent pas de procédure permettant de sélectionner certains points *bien adaptés* à la construction du modèle, ni d'améliorer celui-ci en cas de besoin. C'est une des principales différences entre nos méthodes TR et les codes déjà implémentés que nous utilisons. Si nous avions disposé de plus de temps, nous aurions pu implémenter une telle procédure. Il s'agit d'une amélioration possible pour nos méthodes.

En ce qui concerne la comparaison avec l'algorithme génétique, on constate qu'on peut toujours trouver au moins une méthode locale qui produise un résultat aussi bon ou meilleur. L'algorithme génétique avec Move-Limit (MLGA) permet généralement de déterminer une solution plus précise que l'algorithme génétique seul. Son comportement se rapproche d'ailleurs de celui des méthodes hybrides. Les deux seuls cas où l'algorithme avec Move-Limit est

3. C'est le nombre de runs utilisé dans l'article [29].



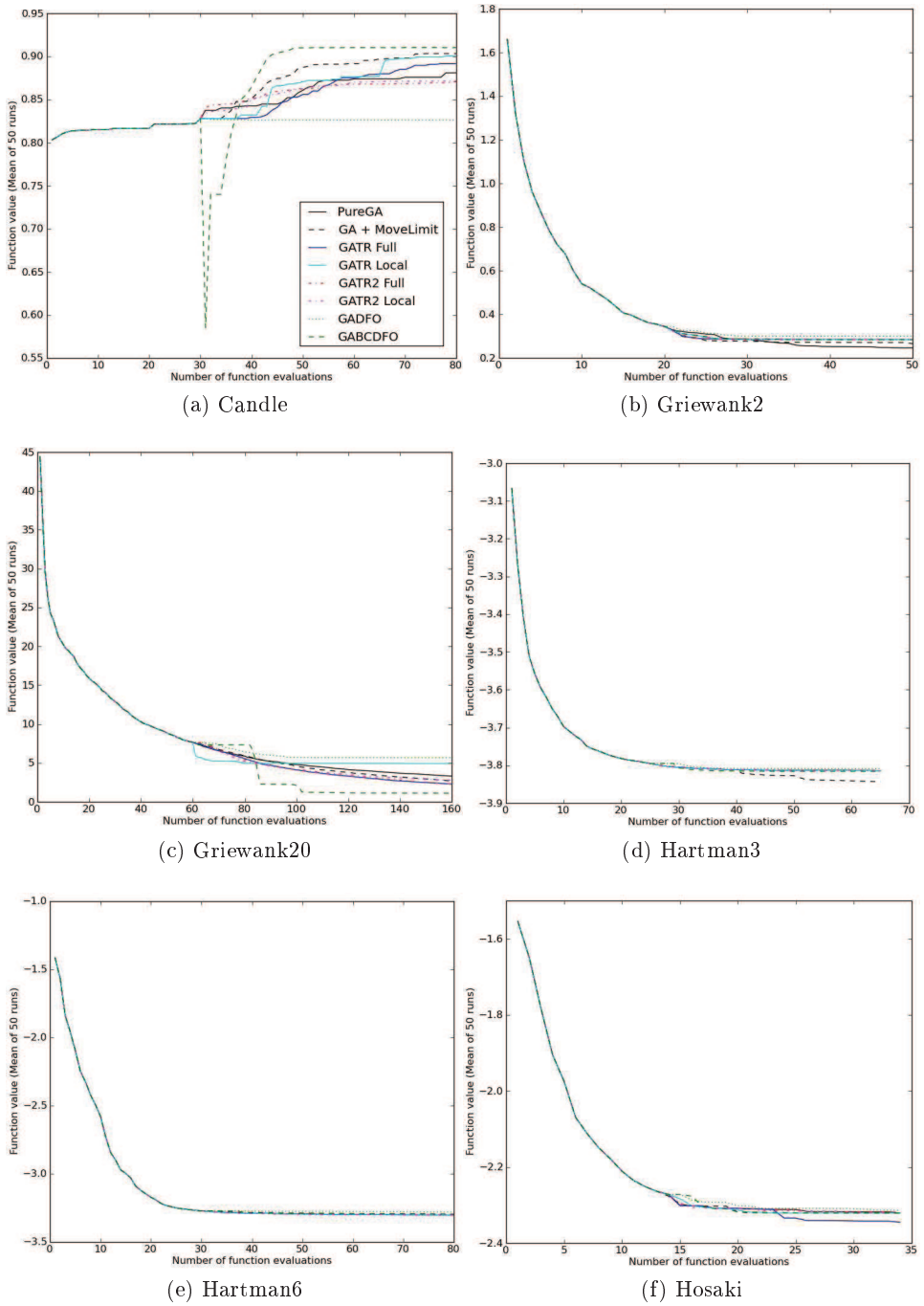
(a) Ackley2 (b) Ackley10

(c) Ackley20 (d) Branin

(e) Branin Modified (f) Camelback

FIGURE 5.17 – Valeur de fonction moyenne en fonction du nombre d'évaluations de fonction.

légèrement moins bon que l'algorithme génétique "pur" sont les fonctions Rastrigin et Griewank à deux dimensions. Il s'agit de fonctions fortement multimodales et l'algorithme génétique pur obtient de meilleurs résultats de par sa plus grande capacité d'exploration de l'espace de recherche.



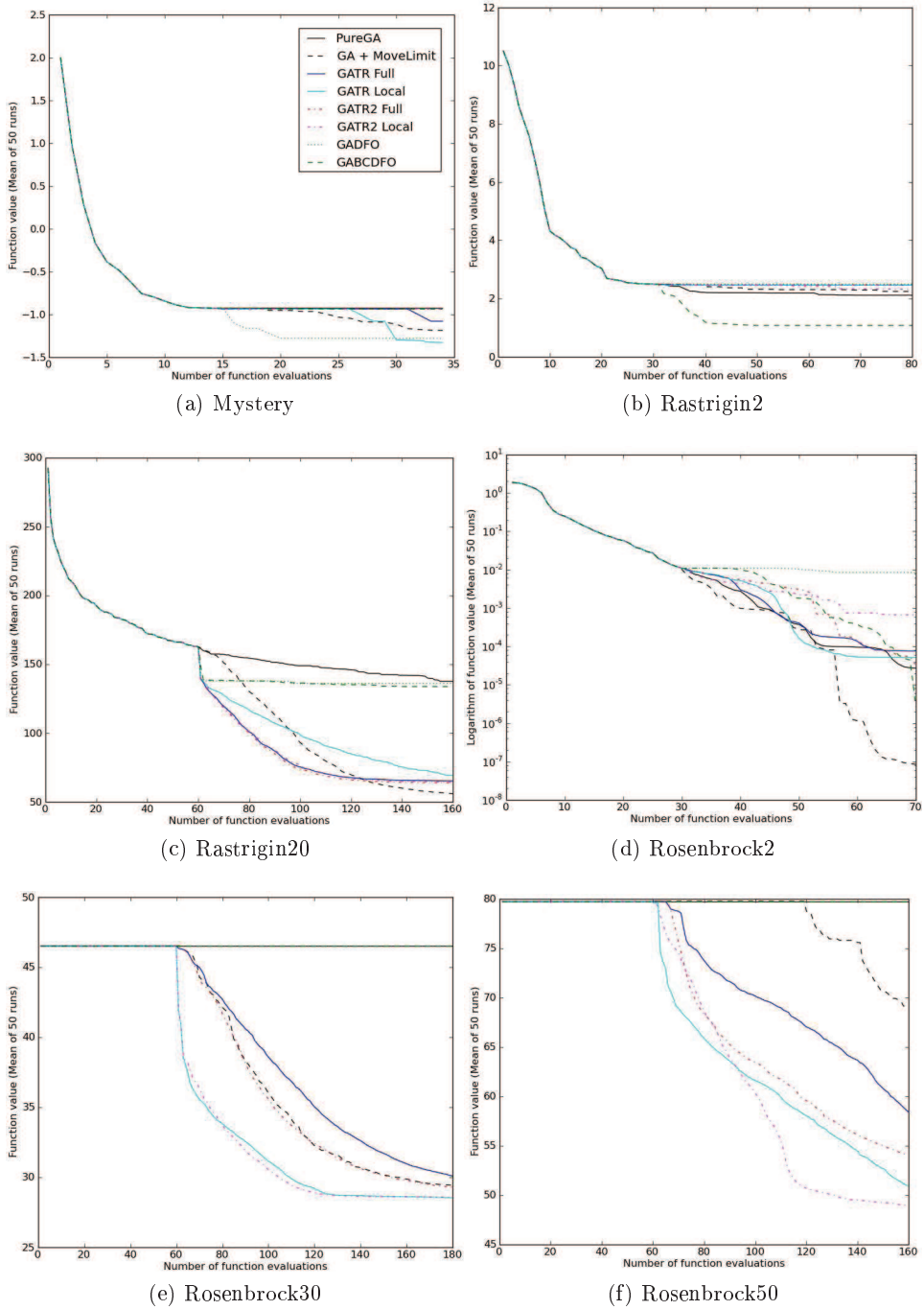
(a) Candle (b) Griewank2

(c) Griewank20 (d) Hartman3

(e) Hartman6 (f) Hosaki

FIGURE 5.18 – Valeur de fonction moyenne en fonction du nombre d'évaluations de fonction (suite).

La différence de comportement entre les méthodes PureGA et MLGA est aussi très visible pour les problèmes de grande dimension. C'est le cas de la fonction Rosenbrock à 30 et 50 variables, où l'algorithme génétique pur ne parvient pas à diminuer la valeur de la fonction objectif. Au contraire,



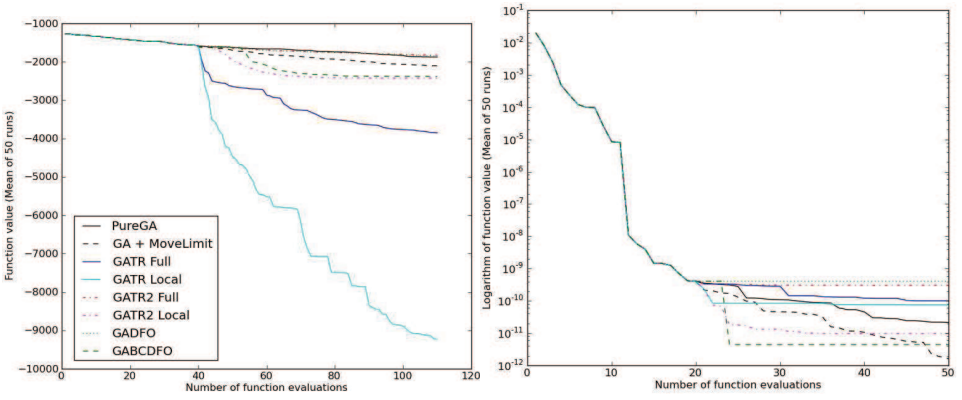
(a) Mystery (b) Rastrigin2

(c) Rastrigin20 (d) Rosenbrock2

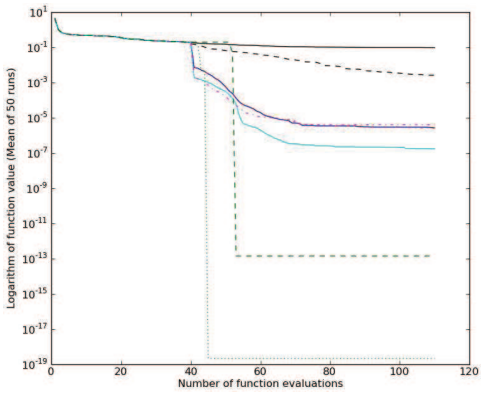
(e) Rosenbrock30 (f) Rosenbrock50

FIGURE 5.19 – Valeur de fonction moyenne en fonction du nombre d'évaluations de fonction (suite).

on constate qu'une fois l'option de Move-Limit activée, la fonction objectif prend des valeurs plus petites. Pour les fonctions Ackley (10 et 20 variables) et Rastrigin (20 variables), la différence entre PureGA et MLGA est toujours très marquée, même si pour ces problèmes l'algorithme génétique seul permet



(a) Schwefel710 (b) Sphere2



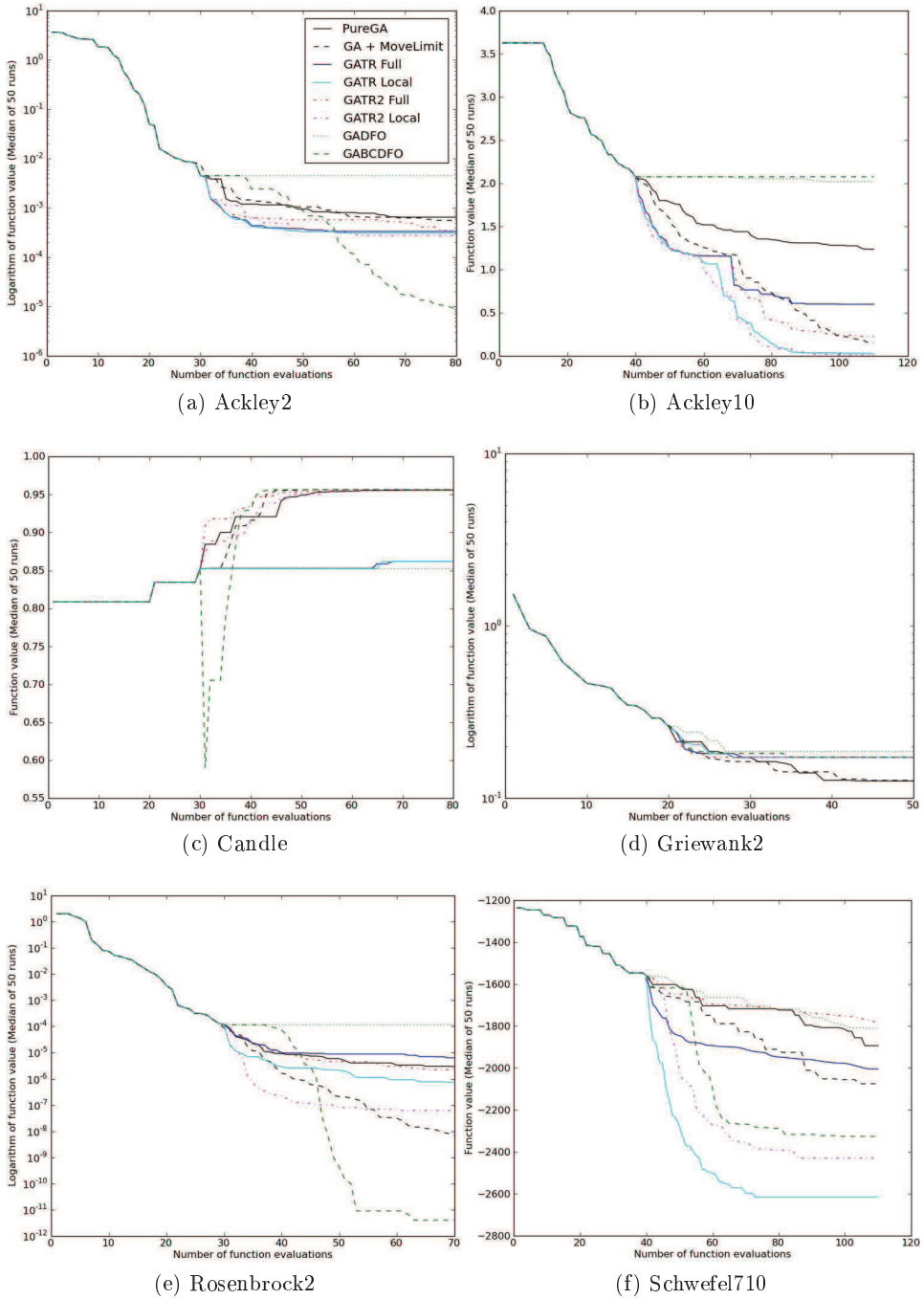
(c) Sphere10

FIGURE 5.20 – Valeur de fonction moyenne en fonction du nombre d'évaluations de fonction (fin).

quand même d'obtenir une diminution dans la fonction objectif.

Pour les problèmes de grande dimension, on constate que nos méthodes TR permettent d'obtenir de meilleures solutions que les méthodes GADFO et GABCDFO. Cela peut s'expliquer par une plus grande difficulté pour ces méthodes de gérer des problèmes possédant de nombreuses variables. Comme nous l'avons signalé à la Section 2.2.3, l'avantage des modèles RBF utilisés dans nos implémentations est qu'ils nécessitent moins de points pour être construits que les approximations (polynômes) utilisés dans DFO et BCDFO. Cela devient un atout non négligeable lorsque le nombre de variables de la fonction étudiée est important.

Durant l'analyse de nos résultats, nous avons également calculé la médiane des valeurs prises par la fonction objectif. L'avantage de cette quantité est qu'elle est moins sensible aux valeurs extrêmes que la moyenne. Nous pouvons ainsi obtenir des courbes aux comportements différents de celles représentant les valeurs moyennes. Nous avons regroupé à la Figure 5.21 les graphiques de ces médianes pour les fonctions pour lesquelles cela donne un autre "classement" des méthodes. C'est le cas, par exemple, de la fonction Ackley à deux variables. Pour cette dernière, les courbes moyennes in-



(a) Ackley2 (b) Ackley10

(c) Candle (d) Griewank2

(e) Rosenbrock2 (f) Schwefel710

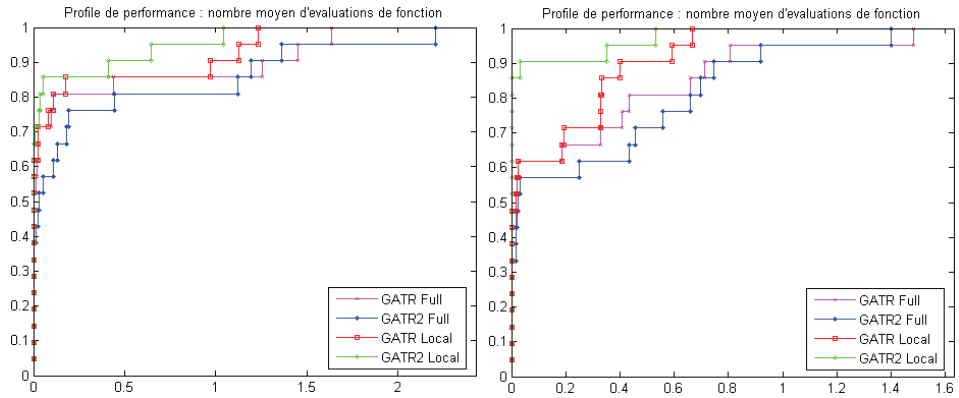
FIGURE 5.21 – Valeur de fonction médiane en fonction du nombre d'évaluations de fonction.

diquent une meilleure performance de l'algorithme génétique par rapport à nos méthodes TR. Cependant, la médiane nous apprend que les méthodes TR donnent de meilleurs résultats. Tracer la médiane pour la fonction Ackley à 10 variables nous montre que les méthodes TR avec un modèle construit

avec un sous ensemble de points (local) produisent de meilleures solutions que les méthodes TR qui prennent en compte tous les points. Le graphique des médianes pour la fonction Rosenbrock (2 dimensions) est intéressant, non pas pour le classement des méthodes mais pour l'échelle de l'ordonnée. A nouveau, on constate que les valeurs moyennes sont supérieures, faussées par quelques résultats moins bons, ce qui ne transparait pas dans les résultats médians.

Pour plus de détails sur ces résultats, les Tableaux 5.9, 5.10, 5.11 et 5.12 placés à la fin de ce chapitre regroupent les valeurs moyennes, médianes, minimales et maximales [4] prises par les 21 fonctions tests, et pour les 8 méthodes testées.

Nous avons ensuite comparé nos méthodes grâce aux profils de performance. Nous disposons de 8 méthodes, ce qui est beaucoup pour obtenir des profils lisibles. C'est pourquoi nous avons d'abord comparé les quatres méthodes TR qui utilisent des modèles RBF et que nous avons implémentées. Pour ce faire, nous utilisons le nombre d'évaluations de fonction nécessaire pour vérifier l'équation (5.3) comme critère de comparaison. Nous avons calculé les profils pour une précision *tol* de 1e-4 et 1e-6. Le résultat est représenté à la Figure 5.22.



(a) *tol* = 1e-4 (b) *tol* = 1e-6

FIGURE 5.22 – Profils de performances pour les algorithmes GATR Full, GATR Local, GATR2 Full et GATR2 Local, réalisés sur les 21 fonctions tests.

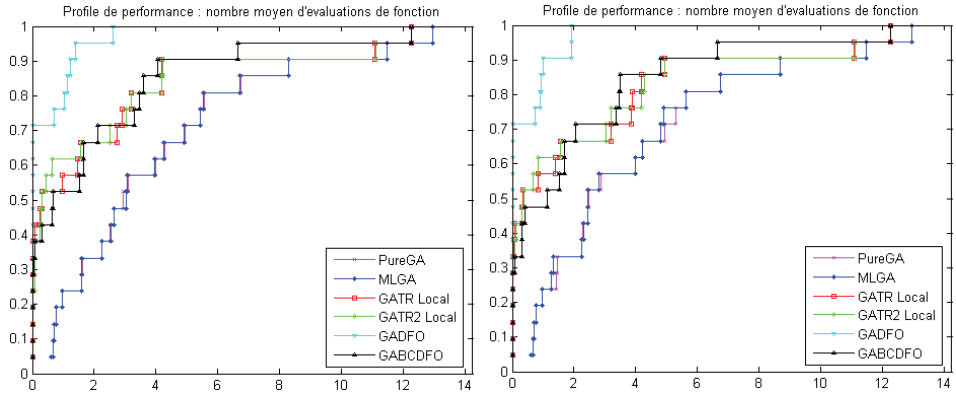
Nous pouvons constater que, dans les deux cas, les méthodes GATR Local et GATR2 Local sont supérieures aux méthodes GATR Full et GATR2 Full. Cela concorde également avec les observations réalisées sur les graphiques des fonctions des Figures 5.17, 5.18, 5.19 et 5.20.

Nous avons ensuite réalisé des profils en conservant ces deux meilleures méthodes (GATR Local et GATR2 Local) auxquelles nous avons ajouté les deux méthodes génétiques (PureGA et MLGA) ainsi que les algorithmes hybrides GADFO et GABCDFO. Nous les traçons toujours dans les mêmes

4. Dans le tableau, cela correspond aux meilleures et pires valeurs suivant qu'on maximise ou minimise la fonction.

conditions que précédemment et nous obtenons les graphiques représentés à la Figure 5.23.

La première observation est la similarité des profils pour les deux précisions. Nous constatons ensuite une distinction entre les algorithmes génétiques et les méthodes hybrides. Les méthodes hybrides se révèlent les plus efficaces. Parmi celles-ci, la méthode GADFO est celle qui se révèle être la meilleure au vu des profils de performance. Les trois autres sont difficilement départageables. Cette supériorité de GADFO pourrait s'expliquer par le fait que l'ensemble de fonctions tests contient plus de problèmes de petite que de grande dimension. Or GADFO s'avère donner de bons résultats pour les fonctions ayant peu de variables, comme nous l'avons vu précédemment avec l'analyse des résultats pour les différents problèmes.



(a) tol = 1e-4 (b) tol = 1e-6

FIGURE 5.23 – Profils de performances pour les algorithmes GATR Local, GATR2 Local, GADFO, BABCDFO, PureGA et GA+Move-Limit, réalisés sur les 21 fonctions tests.

Finalement, un dernier point sur lequel peut s'appuyer la comparaison de nos différentes méthodes est leur temps d'exécution. Nous avons mesuré celui-ci pour chaque run, sur chaque fonction et pour tous les algorithmes. Nous avons ensuite calculé les valeurs moyennes qui sont reprises au Tableau 5.8. On remarque que l'algorithme avec Move-Limit s'exécute plus rapidement que l'algorithme génétique seul. Cela s'explique par le fait que moins de points sont pris en compte par MLGA pour la constrution du modèle, ce qui rend cette dernière méthode plus rapide. La même constatation s'applique entre GATR et GATRS (ainsi que pour GATR2 et GATR2S). La grande différence de temps entre les méthodes TR et TR2 provient de l'utilisation d'IPOPT qui est beaucoup plus lente que le calcul du pas par Steihaug-Toint. Cependant, même si les méthodes utilisant TR2 prennent un peu plus de temps, elles gardent un temps d'exécution comparable à celui de l'algorithme génétique pur (voire plus faible pour GATR2S) tout en produisant des résultats de meilleure qualité. En ce qui concerne les méthodes GADFO et GABCDFO, elles sont très rapides pour les problèmes de petite dimension, pour lesquels elles donnent, rappelons-le, souvent de bons résultats. Elles se révèlent moins adaptées pour des fonctions qui possèdent plus

de 10 variables.

Nous avons testé et développé plusieurs méthodes hybrides. Elles ont toutes révélé leur efficacité. Certaines produiront efficacement des résultats pour des fonctions à peu de variables tandis que d'autres seront utilisées de préférence pour les problèmes de grande dimension. A côté des méthodes hybrides classiques, couplant l'algorithme et une méthode de recherche locale, l'algorithme génétique avec Move-Limit a aussi prouvé son efficacité. L'enjeu d'une bonne optimisation sera de choisir une méthode bien adaptée au type de problème étudié.

	PureGA	MLGA	GADFO	GABCDFO	GATR	GATRS	GATR2	GATR2S
Ackley2	25.15	19.33	4.73	7.10	16.67	10.72	29.68	12.48
Ackley10	61.05	40.65	13.71	19.47	35.67	21.24	101.31	32.37
Ackley20	1169.35	911.28	276.12	286.82	324.86	295.62	478.90	380.57
Branin	7.36	5.49	2.19	3.80	3.86	2.67	10.65	5.70
Branin M.	7.27	5.66	2.17	3.40	3.53	2.82	10.01	4.86
Camelback	3.58	3.00	1.26	2.55	1.92	1.68	4.49	2.85
Candle	25.18	11.30	4.85	6.43	10.87	5.50	36.42	12.68
Griewank2	7.97	7.15	2.16	3.36	4.66	2.48	8.36	5.20
Griewank20	1222.11	1013.40	278.52	289.49	365.04	288.34	441.49	413.90
Hartman3	14.54	9.73	3.24	5.24	8.37	8.38	16.77	7.60
Hartman6	20.58	14.85	5.21	8.84	13.38	13.40	27.32	13.73
Hosaki	3.42	3.13	1.27	2.50	1.71	1.61	4.94	2.79
Mystery	3.41	3.14	1.27	2.54	1.78	1.58	5.18	3.21
Rastrigin2	25.23	16.38	4.80	6.92	16.64	8.14	26.37	12.76
Rastrigin20	1207.63	735.53	577.88	589.46	724.60	615.92	871.96	788.40
Rosenbrock2	13.49	8.85	3.55	5.45	7.88	6.12	17.71	8.38
Rosenbrock30	2027.71	1280.02	426.92	440.86	547.16	464.44	2217.40	729.52
Rosenbrock50	3183.79	1885.10	1846.74	1669.35	1946.91	1728.18	5804.88	2570.46
Schwefel710	88.94	30.10	16.15	20.79	58.55	18.32	71.20	32.00
Sphere2	6.66	5.88	2.07	3.07	3.11	2.65	11.56	7.46
Sphere10	71.40	43.71	14.87	20.26	33.57	22.70	162.81	45.51

TABLE 5.8 – Temps moyens d'exécution (en secondes) des 8 algorithmes sur les 21 problèmes tests.

	MLGA	PureGa	GADFO	GABCDFO	GATR	GA2TR	GATRS	GA2TRS
Ackley2								
moyenne	4.138251E-01	9.150654E-01	1.114687E+00	1.548415E-01	1.179391E+00	1.179436E+00	1.179386E+00	1.152487E+00
médiane	5.577504E-04	6.460223E-04	2.667058E-03	8.944538E-06	3.329229E-04	3.458874E-04	3.067187E-04	2.761565E-04
meilleure	6.460811E-05	6.450319E-05	1.273773E-04	4.440892E-16	2.856562E-05	2.856029E-05	2.630908E-05	6.469164E-06
pire	2.600177E+00	3.574452E+00	3.574452E+00	2.579928E+00	3.574452E+00	3.574452E+00	3.574452E+00	3.574452E+00
Ackley10								
moyenne	4.489676E-01	1.180408E+00	1.960574E+00	2.073973E+00	6.763601E-01	6.368364E-01	6.411323E-01	6.030538E-01
médiane	1.513065E-01	1.233966E+00	1.786289E+00	2.073493E+00	5.979538E-01	2.248681E-01	2.455692E-02	5.841673E-03
meilleure	2.971538E-02	5.203932E-01	8.001330E-01	1.309678E+00	3.754910E-04	6.211357E-04	3.942780E-04	1.032702E-03
pire	2.014157E+00	1.723703E+00	1.008961E+01	2.830779E+00	2.013321E+00	2.013329E+00	2.013316E+00	2.013315E+00
Ackley20								
moyenne	1.266905E+00	1.405208E+00	2.288644E+00	2.288644E+00	1.252851E+00	1.249753E+00	1.270273E+00	1.277822E+00
médiane	1.155925E+00	1.238748E+00	2.270500E+00	2.270500E+00	1.292941E+00	1.292058E+00	1.423538E+00	1.301512E+00
meilleure	8.359803E-02	5.232351E-01	9.514210E-01	9.514210E-01	2.134388E-02	8.874751E-04	1.471017E-02	3.259966E-03
pire	3.574468E+00	3.181681E+00	3.625385E+00	3.625385E+00	3.574464E+00	3.574465E+00	3.574455E+00	3.574455E+00
Branin								
moyenne	4.858392E-01	5.230570E-01	4.203104E-01	5.069885E-01	4.018278E-01	4.419127E-01	4.069500E-01	3.978875E-01
médiane	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01
meilleure	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01	3.978874E-01
pire	1.943140E+00	1.943141E+00	9.013389E-01	1.943141E+00	5.808964E-01	1.052449E+00	8.509878E-01	3.978940E-01
Branin Modified								
moyenne	-1.562198E+01	-1.515260E+01	-1.656269E+01	-1.664402E+01	-1.663106E+01	-1.515356E+01	-1.627981E+01	-1.572834E+01
médiane	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01
meilleure	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01	-1.664402E+01
pire	-7.491700E+00	-7.491700E+00	-1.595504E+01	-1.664402E+01	-1.599588E+01	-7.491700E+00	-1.227624E+01	-7.491700E+00

TABLE 5.9 – Résultats statistiques des 5 premières fonctions tests pour les 8 algorithmes.

	MLGA	PureGa	GADFO	GABCDFO	GATR	GA2TR	GATRS	GA2TRS
Camelback								
moyenne	-1.029131E+00	-1.025239E+00	-9.838515E-01	-9.984208E-01	-9.929219E-01	-9.929216E-01	-9.989817E-01	-9.989747E-01
médiane	-1.031628E+00	-1.031628E+00	-1.031414E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00
meilleure	-1.031628E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00	-1.031628E+00
pire	-9.159483E-01	-7.286315E-01	-2.153293E-01	-2.154638E-01	-2.154637E-01	-2.154633E-01	-2.154638E-01	-2.154638E-01
Candle								
moyenne	9.031942E-01	8.809581E-01	8.708576E-01	9.101192E-01	8.916065E-01	8.705297E-01	9.012129E-01	8.713275E-01
médiane	9.563294E-01	9.558305E-01	9.563101E-01	9.563294E-01	8.617809E-01	9.563283E-01	8.616823E-01	9.561046E-01
meilleure	1.000000E+00	9.563294E-01	9.999993E-01	1.000000E+00	9.563294E-01	9.999981E-01	9.563294E-01	9.999712E-01
pire	6.990762E-01	6.959143E-01	6.989679E-01	6.990762E-01	8.136128E-01	6.990732E-01	8.136128E-01	6.981462E-01
Griewank2								
moyenne	2.647277E-01	2.453094E-01	2.891132E-01	2.827530E-01	2.844815E-01	2.838481E-01	2.836544E-01	2.827530E-01
médiane	1.265197E-01	1.257196E-01	1.727421E-01	1.726071E-01	1.726245E-01	1.726096E-01	1.726071E-01	1.726071E-01
meilleure	7.396041E-03	7.620260E-03	1.779644E-02	7.396040E-03	7.396044E-03	7.396134E-03	7.396040E-03	7.396040E-03
pire	2.769306E+00	2.769307E+00	2.769306E+00	2.769306E+00	2.769307E+00	2.769306E+00	2.769306E+00	2.769306E+00
Griewank20								
moyenne	2.676287E+00	3.293803E+00	5.126904E+00	1.104894E+00	2.280744E+00	2.281988E+00	4.934054E+00	2.756504E+00
médiane	2.541612E+00	3.247911E+00	5.183751E+00	1.021523E+00	2.243329E+00	2.274005E+00	4.532612E+00	1.829216E+00
meilleure	2.137772E+00	2.570012E+00	2.446577E+00	1.741045E-02	1.688896E+00	1.710247E+00	1.467588E+00	1.259352E+00
pire	3.692516E+00	4.803934E+00	8.862395E+00	2.610109E+00	3.527975E+00	3.287520E+00	1.057825E+01	9.719907E+00
Hartman3								
moyenne	-3.843143E+00	-3.814202E+00	-3.809027E+00	-3.816401E+00	-3.815004E+00	-3.814989E+00	-3.815004E+00	-3.814969E+00
médiane	-3.862782E+00	-3.862782E+00	-3.862471E+00	-3.862782E+00	-3.862782E+00	-3.862781E+00	-3.862782E+00	-3.862782E+00
meilleure	-3.862782E+00	-3.862782E+00	-3.862782E+00	-3.862782E+00	-3.862782E+00	-3.862782E+00	-3.862782E+00	-3.862782E+00
pire	-3.089764E+00	-3.089764E+00	-3.089763E+00	-3.089764E+00	-3.089764E+00	-3.089764E+00	-3.089764E+00	-3.089764E+00

TABLE 5.10 – Résultats statistiques des 5 fonctions tests suivantes pour les 8 algorithmes.

	MLGA	PureGa	GADFO	GABCDFO	GATR	GA2TR	GATRS	GA2TRS
Hartman6								
moyenne	-3.297760E+00	-3.301338E+00	-3.280897E+00	-3.295339E+00	-3.302493E+00	-3.302634E+00	-3.189640E+00	-3.300990E+00
médiane	-3.335391E+00	-3.335364E+00	-3.328858E+00	-3.335392E+00	-3.335392E+00	-3.335391E+00	-3.335392E+00	-3.335392E+00
meilleure	-3.335392E+00	-3.335391E+00	-3.335303E+00	-3.335392E+00	-3.335392E+00	-3.335392E+00	-3.335392E+00	-3.335392E+00
pire	-3.092128E+00	-3.164030E+00	-3.052820E+00	-3.106074E+00	-3.182731E+00	-3.189640E+00	-3.182731E+00	-3.141384E+00
Hosaki								
moyenne	-2.321451E+00	-2.321276E+00	-2.313756E+00	-2.321451E+00	-2.345353E+00	-2.320993E+00	-2.321450E+00	-2.321444E+00
médiane	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00
meilleure	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00	-2.345812E+00
pire	-1.127794E+00	-1.127794E+00	-1.127776E+00	-1.127794E+00	-2.331639E+00	-1.127794E+00	-1.127794E+00	-1.127794E+00
Mystery								
moyenne	-1.186834E+00	-9.283281E-01	-9.356843E-01	-9.377939E-01	-1.080246E+00	-9.377942E-01	-1.330240E+00	-9.377966E-01
médiane	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00
meilleure	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00	-1.456526E+00
pire	2.866218E+00	2.866218E+00	2.866221E+00	2.866218E+00	2.866218E+00	2.866218E+00	2.866218E+00	2.866218E+00
Rastrigin2								
moyenne	2.242570E+00	2.111244E+00	2.435189E+00	1.074555E+00	2.467496E+00	2.467496E+00	2.467496E+00	2.320433E+00
médiane	1.989918E+00	1.989918E+00	1.989920E+00	9.949591E-01	1.989918E+00	1.989918E+00	1.989918E+00	1.989918E+00
meilleure	4.789058E-12	3.607017E-10	6.186929E-04	0.000000E+00	3.563709E-10	3.560263E-10	4.167926E-09	3.337597E-11
pire	9.949560E+00	4.974790E+00	1.010086E+01	4.974790E+00	9.949560E+00	9.949560E+00	9.949561E+00	9.949560E+00
Rastrigin20								
moyenne	5.596810E+01	1.376210E+02	1.463881E+02	1.337838E+02	6.514466E+01	6.374957E+01	6.910873E+01	6.425661E+01
médiane	5.519793E+01	1.429631E+02	1.505697E+02	1.348120E+02	5.721180E+01	5.721172E+01	5.944220E+01	5.721666E+01
meilleure	3.709841E+01	6.243864E+01	6.492210E+01	7.154384E+01	3.486154E+01	3.484899E+01	3.780899E+01	3.780842E+01
pire	8.122922E+01	1.944960E+02	1.959497E+02	1.728581E+02	1.548663E+02	1.548643E+02	1.645929E+02	1.529524E+02

TABLE 5.11 – Résultats statistiques des 5 fonctions tests suivantes pour les algorithmes.

	MLGA	PureGa	GADFO	GABCDFO	GATR	GA2TR	GATRS	GA2TRS
Rosenbrock2								
moyenne	8.411439E-08	2.695191E-05	5.112156E-03	3.137967E-07	7.603359E-05	3.224836E-05	5.211885E-05	6.641400E-04
médiane	6.807440E-09	2.934080E-06	1.769196E-05	4.000000E-12	6.385938E-06	2.248968E-06	7.242499E-07	6.151388E-08
meilleure	7.006960E-11	3.997197E-09	1.046870E-09	0.000000E+00	1.516462E-08	7.916637E-08	3.580054E-11	6.111749E-12
pire	8.515121E-07	5.319716E-04	7.921261E-02	1.050530E-05	1.679173E-03	4.623162E-04	2.116462E-03	2.022916E-02
Rosenbrock30								
moyenne	2.942250E+01	4.652000E+01	4.652000E+01	4.652000E+01	3.007635E+01	2.927830E+01	2.854057E+01	2.856912E+01
médiane	2.907067E+01	4.652000E+01	4.652000E+01	4.652000E+01	2.992249E+01	2.924477E+01	2.857876E+01	2.860876E+01
meilleure	2.854262E+01	4.652000E+01	4.652000E+01	4.652000E+01	2.879539E+01	2.874425E+01	2.762447E+01	2.798747E+01
pire	3.126442E+01	4.652000E+01	4.652000E+01	4.652000E+01	3.341160E+01	3.197205E+01	2.886894E+01	2.883190E+01
Rosenbrock50								
moyenne	6.887062E+01	7.976000E+01	7.976000E+01	7.976000E+01	5.842947E+01	5.398662E+01	5.091722E+01	4.896870E+01
médiane	6.514465E+01	7.976000E+01	7.976000E+01	7.976000E+01	5.812528E+01	5.434851E+01	5.092044E+01	4.867981E+01
meilleure	5.789713E+01	7.976000E+01	7.976000E+01	7.976000E+01	5.586067E+01	4.911269E+01	4.879344E+01	4.838455E+01
pire	7.976000E+01	7.976000E+01	7.976000E+01	7.976000E+01	6.329921E+01	5.867462E+01	5.529498E+01	5.086482E+01
Schwefel710								
moyenne	-2.104601E+03	-1.875401E+03	-1.847585E+03	-2.377292E+03	-3.853460E+03	-1.829679E+03	-9.228151E+03	-2.426120E+03
médiane	-2.075487E+03	-1.893618E+03	-1.826057E+03	-2.325896E+03	-2.004586E+03	-1.782799E+03	-2.615242E+03	-2.430582E+03
meilleure	-3.003492E+03	-2.975313E+03	-2.864448E+03	-3.281074E+03	-3.849504E+04	-2.722760E+03	-4.301995E+04	-3.357726E+03
pire	-1.217654E+03	-1.299046E+03	-1.164574E+03	-1.869293E+03	-1.204411E+03	-1.168082E+03	-1.464923E+03	-1.209760E+03
Sphere2								
moyenne	1.623188E-12	2.135195E-11	4.069309E-10	4.426063E-12	1.001735E-10	3.045416E-10	7.443613E-11	9.778014E-12
médiane	8.049350E-13	5.318607E-12	7.425108E-11	0.000000E+00	9.516042E-12	1.746774E-11	3.352127E-12	3.097249E-12
meilleure	3.027459E-15	2.327347E-14	2.165825E-13	0.000000E+00	2.165825E-13	2.165825E-13	9.409397E-14	2.529077E-14
pire	9.312355E-12	2.645500E-10	7.090449E-09	3.348124E-11	1.892974E-09	7.090449E-09	2.465868E-09	7.016280E-11
Sphere10								
moyenne	2.623174E-03	9.593840E-02	2.179171E-19	1.400000E-13	2.675916E-06	2.580978E-06	1.693386E-07	4.028526E-06
médiane	1.228355E-03	8.728281E-02	2.124698E-19	0.000000E+00	1.114915E-06	8.450066E-07	6.045112E-08	3.907151E-09
meilleure	3.487021E-05	3.055603E-02	6.814559E-20	0.000000E+00	2.202342E-07	1.302582E-07	1.427453E-09	8.385245E-11
pire	2.088817E-02	1.967124E-01	4.247784E-19	1.000000E-12	1.700758E-05	1.380657E-05	1.259848E-06	9.888544E-05

TABLE 5.12 – Résultats statistiques des 6 dernières fonctions tests pour les 8 algorithmes.

Conclusion et Perspectives

Au cours de ce mémoire, nous avons développé et testé des méthodes hybrides. Celles-ci combinent un algorithme génétique assisté par modèles et des méthodes de recherche locale de type région de confiance. La première étape de notre recherche a été de consulter la littérature afin de déterminer ce qui avait déjà été réalisé en terme de couplage entre algorithme génétique et méthode locale. Nous nous sommes basés sur les articles de Tenne et Armfield [28, 29, 30] pour construire la structure de nos algorithmes hybrides. Ces articles nous ont également incités à utiliser des modèles RBF de la fonction objectif, d'autant plus que les modèles RBF sont déjà exploités avec succès dans Minamo.

Dans un second temps, nous avons développé et implémenté deux méthodes de région de confiance qui font intervenir ces modèles RBF. Nous avons couplé ces méthodes à l'algorithme génétique de Minamo afin de construire des méthodes hybrides. Nous avons également couplé deux autres méthodes de région de confiance, que nous n'avons pas implémentées mais dont nous avons utilisé les codes existants. L'implémentation en C++ et Python, ainsi que le couplage ont été réalisés dans le logiciel Minamo de l'entreprise Cenaero. L'implémentation des méthodes fait appel à de nombreuses fonctions déjà implémentées dans le logiciel et a nécessité la modification de plusieurs fonctions déjà implémentées. Les méthodes locales implémentées ne sont donc pas indépendantes des autres codes de Minamo. Pour des raisons de confidentialité, nous ne pouvons donc pas dévoiler les codes que nous avons implémentés dans ce mémoire.

Après la phase d'implémentation, nous avons testé les méthodes hybrides ainsi construites sur un ensemble de fonctions mathématiques. Notre but était de comparer les performances des différentes méthodes pour un nombre d'évaluations de la fonction objectif fixé. Ces tests nous ont permis de dégager différentes conclusions. La comparaison de nos méthodes hybrides avec l'algorithme génétique où l'option du Move-Limit est activée nous permet de constater l'efficacité de cette option pour imiter une méthode locale. Cependant, la convergence observée pour les méthodes hybrides que nous avons implémentées est plus rapide que celle de l'algorithme génétique avec Move-Limit. Dans la plupart des cas, on constate que l'algorithme génétique sans Move-Limit produit de moins bons résultats que toutes les autres méthodes. Cette observation est encore plus marquée pour les fonctions qui possèdent beaucoup de variables. En ce qui concerne les méthodes hybrides, nous ne pouvons pas en désigner une qui soit plus performante que les autres. Certaines, telles celles utilisant DFO et BCDFO, produisent d'excellents résul-

tats pour les problèmes de petite dimension. D'autres comme celles utilisant les méthodes de région de confiance avec un modèle RBF, permettent d'obtenir une meilleure solution pour les fonctions de grande dimension. Suivant le problème que nous désirons résoudre, il sera donc conseillé de choisir un algorithme hybride du premier type ou du second.

Ce mémoire constitue une première étape dans le développement de méthodes hybrides basées sur des régions de confiance à Cenaero. De nombreuses perspectives s'ouvrent pour de futures recherches dans la continuité de ce travail. Lors des différents tests, nous avons constaté que la définition actuelle du RBF dans Minamo était plus adaptée à l'algorithme génétique qu'à un modèle pour une région de confiance. Il serait intéressant d'inclure dans Minamo des procédures pour sélectionner les points qui interviennent dans la construction du modèle RBF et améliorer la qualité du modèle lorsque cela s'avère nécessaire. Un schéma similaire à la quatrième étape de l'algorithme DFO (Algorithme 4.4) pourrait être envisagé.

Une fois que les algorithmes hybrides ont été validés grâce aux tests sur les fonctions mathématiques, l'étape suivante est de réaliser des tests sur des cas réels. Nous n'avons pas pu réaliser de tels tests dans le cadre de ce mémoire. En effet, ces tests prennent beaucoup de temps. En outre, nous préférons assurer une bonne stabilité des nouvelles méthodes avant de les appliquer sur un cas réel. Appliquer trop rapidement les méthodes sur les problèmes réels peut mener à des résultats biaisés si nous ne sommes pas suffisamment certains de nos implémentations. De plus, nos méthodes sont adaptées pour des problèmes présentant uniquement des contraintes de bornes. Un cas test sans contrainte aurait donc été nécessaire à ce stade. La dernière remarque nous amène à une autre perspective, développer des méthodes hybrides qui permettent de gérer les contraintes de tous types et pas seulement les contraintes de bornes.

Finalement, une dernière perspective est de considérer une alternance entre l'algorithme génétique et la méthode locale au sein de la méthode hybride. Pour équilibrer la recherche entre les deux méthodes, le critère de Métropolis présenté à la Section 3.2.2 pourrait être intéressant.

Bibliographie

[1] Affenzeller M., Beham A., Wagner S., Winkler S., *Genetic Algorithms and Genetic Programming Modern Concepts and practical Applications*, CRC Press, 2009.

[2] Bierlaire M., *Introduction à l'optimisation différentiable*, Presses polytechniques et universitaires romandes, Lausanne, 2006.

[3] Bonnans J. F., Gilbert J. C., Lemaréchal C., Sagastizabal C. A., *Numerical Optimization*, Springer-Verlag, Berlin, 2003.

[4] Conn A. R., Gould N. I. M., Toint P. L., *LANCELOT : a Fortran package for large-scale nonlinear optimization (Release A)*, Number 17 in Springer Series in Computational Mathematics, Springer Verlag, Heidelberg, Berlin, New-York, 1992.

[5] Conn A. R., Gould N. I. M., Toint P. L., *Trust-Region Methods*, Nr 01 in the MPS-SIAM Series on Optimization, SIAM, Philadelphia, USA, 2000.

[6] Conn A. R., Scheinberg K., Vicente L. N., *Introduction to derivative-free optimization*, MPS-SIAM Series on Optimization, SIAM, Philadelphia, USA, 2009.

[7] Dennis J. Z. JR., Schnabel R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[8] Fletcher R., *Practical Methods of Optimization*, Second Edition, John Wiley & Sons, Chichester, 2000.

[9] Griva I., Nash S. G., Sofer A., *Linear and Nonlinear Optimization*, second edition, George Mason University, Fairfax, Virginia, 2009.

[10] Haupt R.L., Haupt S.E., *Practical Genetic Algorithms*, Second Edition, John Wiley & Sons, Hoboken, New Jersey, 2004.

[11] Nocedal J., Wright S. J., *Numerical Optimization*, second edition, Springer, USA, 2006.

[12] Törn A., Zilinskas A., *Global Optimization*, Springer-Verlag, Berlin, 1989.

[13] Leclercq J. P., *Optimisation combinatoire et discrète*, cours, FUNDP, Namur, 2010.

[14] Strodiot J. J., *Optimisation et contrôle*, cours, FUNDP, Namur, 2008.

[15] Toint P., *Algorithmes avancés en optimisation numérique*, cours, FUNDP, Namur, 2010.

[16] Baetmans F., *Etude et implementation des machines à vecteurs de support*, Aout 2010.

[17] Conn A. R., Scheinberg K. and Toint Ph. L., *Recent progress in unconstrained nonlinear optimization without derivatives*, Mathematical Programming, Vol. 79 (1997), pp. 397 - 414.

[18] Conn A. R., Scheinberg K. and Toint Ph. L., *On the convergence of derivative-free methods for unconstrained optimization*, Approximation Theory and Optimization : Tributes to M. J. D. Powell , Eds. A. Iserles and M. Buhmann, (1997), pp. 83-108, Cambridge University Press.

[19] Conn A. R., Scheinberg K. and Toint Ph. L., *A derivative free optimization algorithm in practice*, in Proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO, 1998.

[20] Dolan E. D., Moré J. J., *Benchmarking optimization software with performance profiles*, Springer-Verlag, 2001.

[21] El-Mihoub T. A., Hopgoog A. A., Nolle L., Battersby A., *Hybrid Genetic Algorithms : A Review*, Engineering Letters, 13 :2, EL_13_2_11, 2006.

[22] Forrester A I J, Keane, A J , *Recent advances in surrogate-based optimization*, Progress in Aerospace Sciences, Volume 45, Issues 1-3, Pages 50-79.

[23] Gratton S., Toint Ph. L., Tröltzsch A., *An active-set trust-region method for derivative-free nonlinear bound-constrained optimization*, Report NAXYS-01-2010, 9 July 2010.

[24] More J. J., Wild S. M., *Benchmarking derivative-free optimization algorithms.*

[25] Oeuvray R., *Trust-Region Methods Based on Radial Basis Functions with Application to Biomedical Imaging*, PhD thesis, Institu de Mathématiques, Ecole Polytechnique Fédérale de Lausanne, Suisse, 2005.

[26] Sainvitu C., Iliopoulou V., Lepot I., *Global Optimization with expensive Functions - Sample Turbomachinery Design Application.*

[27] Sauer T., XU Y., *On Multivariate Lagrange Interpolation*, Mathematics of computation, 64(211) :1147-1170, 1195.

[28] Tenne Y., Armfield S. W., *A framework for memetic optimization using variable global and local surrogate models*, Springer-Verlag, published online : 22 August 2008.

[29] Tenne Y., Armfield S. W., *A Memetic Algorithm Assisted by an Adaptative Topology RBF Network and Variable Local Models for Expensive Optimization Problems*, in *Advances in Evolutionary Algorithms*, Witold Kosinski, Vienne, 2008.

[30] Tenne Y., Armfield S. W., *A Versatile Surrogate-Assisted Memetic Algorithm for Optimization of Computationally Expensive Functionsd and its Engineering Applications*, Studies in Computational Intelligence (SCI) 92, 43-72,Springer-Verlag, Berlin Heidelberg, 2008.

[31] Torczon V., Trosset M. W., *Using approximations to accelerate engineering design optimization*, AIAA-98-4800 in the Proceedings of the 7th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization,1998.

[32] Wächter A. and Biegler L. T., *On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming*, Mathematical Programming 106(1), pp. 25-57, 2006.

[33] Alliot J. M., *Les algorithmes génétiques*, `http://recherche.enac.fr/opti/papers/thesis/HABIT/main002.html`, consultation le 25 avril 2010.

[34] Obitko M., *Introduction to genetic algorithms*, `http://www.obitko.com/tutorials/genetic-algorithms/index.php`, 1998, consultation le 25 avril 2010.

[35] `http://www.cenaero.be/`

[36] *Cenaero - Annual Report 2009*, `http://cms.horus.be/files/99936/MediaArchive/pdf/Cenaero_Activity_Report_09.pdf`

[37] `https://projects.coin-or.org/Ipopt`

[38] `https://projects.coin-or.org/Dfo`

[39] `http://www.mcs.anl.gov/~more/cops/perf.m`

[40] `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm`

[41] `http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf`

[42] `http://extreme.adorio-research.org/download/mvf/html/mvf.html`